Java

1 Programación orientada a objetos en lenguaje Java

- Introducción al lenguaje Java
- Tipos de datos básicos e instrucciones
- Clases y objetos
- Errores de ejecución y excepciones
- Encapsulación
- Colecciones
- Herencia y polimorfismo
- Clases abstractas e interfaces
- Programación gráfica

IEEE Spectrum Top Programming Languages 2023: lenguajes en ingeniería por tipo de aplicación
https://spectrum.ieee.org/top-programming-languages-2023

PYPL PopularitY of Programming Language: búsquedas de tutoriales en Google
http://pypl.github.io/PYPL.html

Stack Overflow Developer Survey 2023
https://survey.stackoverflow.co/2023/

Linkedin most in-demand skills
https://www.linkedin.com/business/talent/blog/talent-strategy/linkedin-most-in-demand-hard-and-soft-skills

Top IDE index: los sistemas de desarrollo más descargados desde búsquedas en Google
http://pypl.github.io/IDE.html

2 Bibliografía

The Java Language Specification. Java SE 21 Edition.
https://docs.oracle.com/javase/specs/jls/se21/html/index.html

Oracle Java Tutorials
http://docs.oracle.com/javase/tutorial/

tutorialspoint Java basic syntax
http://www.tutorialspoint.com/java/java_basic_syntax.htm

W3 Schools Java Tutorial
https://www.w3schools.com/java/

H.D. Assumpção: Getting started with Intellij Idea

B. Baesens, etc: Beginning Java programming

M.R. Brzustowicz: Data Science with Java

F. Cheng: Exploring Java 9

I.F. Darwin: Java Cookbook

N. Dale, etc: Object-oriented data structures using Java

C. Dea, etc: JavaFX 9 by example

J. Friesen: Learn Java for Android development

M.T. Goodrich, etc: Data structures & algorithms in Java

J. Graba: An introduction to network programming with Java. 3rd edition

J. Juneau, etc: Java 9 recipes

Y.D. Liang: Introduction to Java programming

R. Liguori, etc: Java 8 pocket guide

K. Sharan: Beginning Java 8 Fundamentals

K. Sharan: Beginning Java 8 language features

K. Sharan: Learning JavaFX 8

P. Verhas: Java 9 programming by example

3 Generación de aplicaciones y ejecución

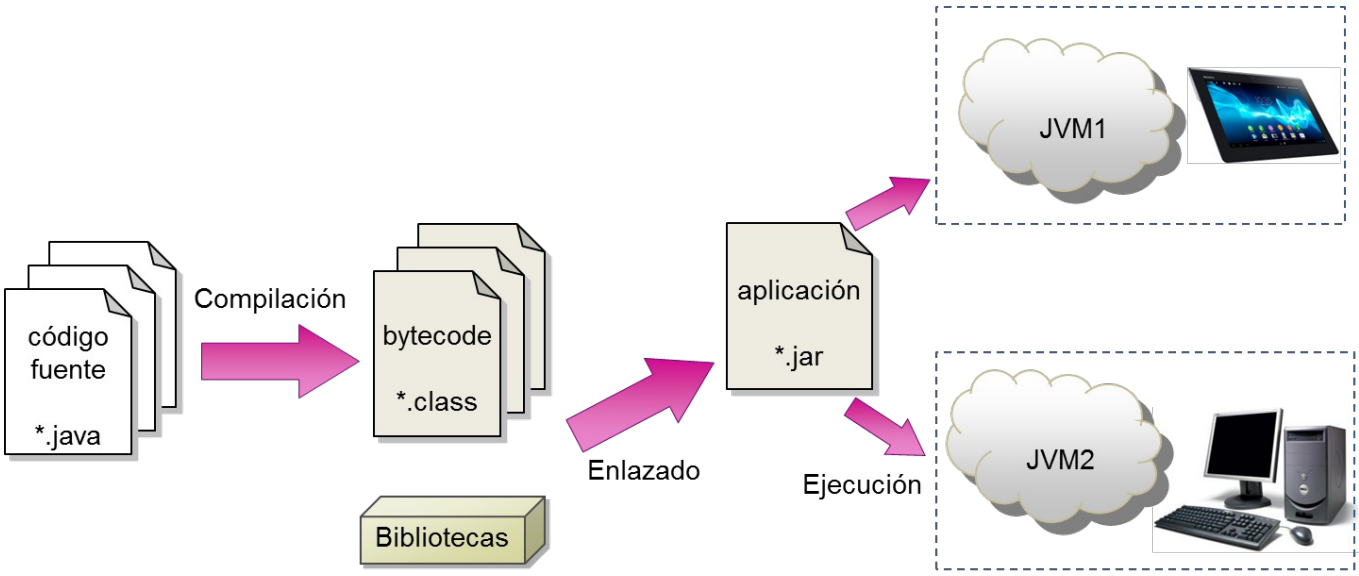
Diseñado para la generación de aplicaciones que no dependen de sistemas operativos ni hardware
WORE - *Write Once Run Everywhere*

Desarrollo de aplicaciones para la web, aplicaciones para escritorio, para dispositivos móviles

Java Platform:

- *Java Virtual Machine* (JVM): máquina virtual que traduce y ejecuta el bytecode

- *Java Application Programming Interface* (Java API): bibliotecas de clases del sistema



3.1 JDK SE - Java Development Kit Standard Edition

Java 21 es la última versión LTS (*Long Term Support*), con actualizaciones gratuitas hasta septiembre de 2026.

Versión comercial: con parches de seguridad futuros, gratuíto para desarrollo y pruebas

https://www.oracle.com/java/technologies/downloads/

Versión de software libre: OpenJDK 21

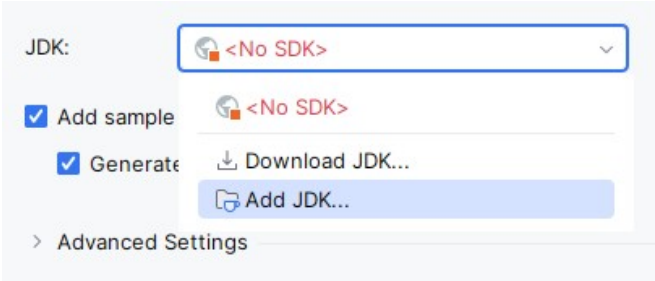
https://jdk.java.net/21/

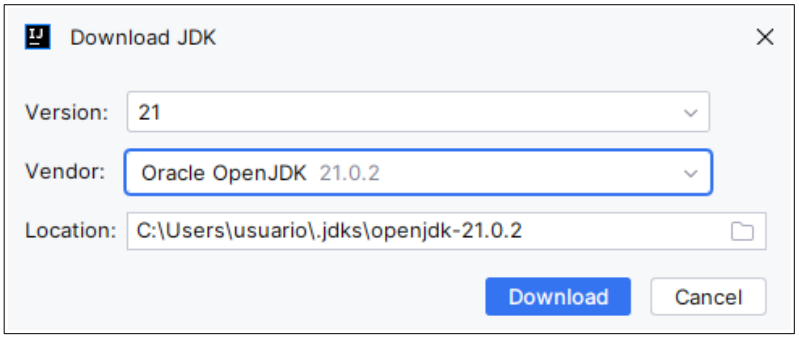
4 Integrated Development Environment (IDE)

Intellij IDEA Community Edition https://www.jetbrains.com/idea/

Entorno gráfico para edición - compilación - depuración de programas

Para la descarga de un Java JDK, en la creación del primer proyecto:





5 Comentarios

```
// Dos barras para indicar comentario hasta salto de línea


/* Comentario delimitado entre barra asterisco hasta asterisco barra
   para insertar comentario que puede ocupar múltiples líneas */
```

La barra y asterisco se puede utilizar dentro de una instrucción para anular parte de ella:

```
int x, y;  // Dos variables enteras
y = 3;  // Asigna un valor a y
x = y + /* 4 */ 5;  // x es y+5, el comentario anula parte de la instrucción
```

6 Tipos de datos básicos

6.1 Números enteros

Tipo	Tamaño	Rango
`byte`	8 bits con signo	$-2^7 = -128$ a $2^7-1 = 127$
`short`	16 bits con signo	$-2^{15} = -32767$ a $2^{15}-1 = 32768$
`int`	32 bits con signo	$-2^{31} = -2147483648$ a $2^{31}-1 = 2147483647$
`long`	64 bits con signo	$-2^{63} = -9223372036854775808$ a $2^{63}-1 = 9223372036854775808$
`BigInteger`	formatos mayores	sin límite

- Conversión automática cuando se amplía el formato:

```
int x = 123;
long y = x;
```

- Casting necesario cuando se reduce de formato:

```
long y = 123;
int x = (int) y;
```

- Constantes

  - En decimal si comienzan por un dígito 1 ... 9

```
int x = 29;  // Expresado en decimal
```

  - En binario si comienzan por 0b

```
int x = 0b11101;  // Expresado en binario
```

  - En octal si comienzan por un 0 y continúa con otro dígito numérico

```
int x = 035;  // Expresado en octal. En binario: 011 101
```

  - En hexadecimal si comienzan por 0x

```
int x = 0x1D;  // Expresado en hexadecimal. En binario: 0001 1101
```

  - Para constantes que exceden el formato `int`, hay que añadir `L`

```
long y = 123456789012345678L;
```

6.2 Números reales

Tipo	Tamaño	Rango
`float`	32 bits	±1.5E-45 a ±3.4E38
`double`	64 bits	±5.0E-324 a ±1.7E308

Constantes

- Por defecto, `double`. Ejemplo: `double a = 123.5;`

- Añadir `f` para `float`. Ejemplo: `float b = 123.5f;`

Representación de los números reales

Formato IEEE 754 de precisión simple=`float` o doble=`double`

En decimal: `signo * M * 2^E`

En formato de precisión simple de 32 bits: `sEEEEEEEEMMMMMMMMMMMMMMMMMMMMMMM`

Bit de signo `s`. Si `s=1`, entonces `signo=-1`. Si `s=0`, entonces `signo=1`.

Mantisa `M` de 23 bits.

Exponente `E` de 8 bits. Al exponente de rango -126 a 127 se le suma 127 para almacenarlo en el rango 1 a 254.

Ejemplo: `float x = -321.6875;`

Es negativo, entonces `signo = 1`

$321.6875 = 2^8 + 2^6 + 2^0 + 2^{-1} + 2^{-3} + 2^{-4} = 101000001.1011$ en binario

El punto decimal se desplaza hacia la izquierda hasta dejar sólo a un 1 en la parte entera:

$101000001.1011 = 1.010000011011 * 2^8$

`M = 010000011011`, que se completa hasta 23 bits `M = 01000001101100000000000`

Exponente `E = 8 + 127 = 135 = 10000111` en binario

Representación en formato IEEE 754: `11000011101000001101100000000000`

Valores especiales

Inifinito: E=255, M=0

Ejemplo: división por cero

```
float x, y, z;  // Declara las variables x, y, z como números reales de simple precisión
x = 7.4f;  // Asigna el valor 7.4 a la variable x
y = 0.0f;  // Asigna el valor 0.0 a la variable y
z = x / y;  // Asigna el resultado de la división x/y a la variable z
```

NaN (Not a Number): E=255, M≠0

Ejemplo: raices cuadradas de números negativos

```
double x, y;  // Declara las variables x, y como números reales de doble precisión

x = -3.0;  // Asigna a x el valor -3.0
y = Math.sqrt(x);  // Asigna a y el resultado de la raíz cuadrada de x
```

Valores especiales en clase `Double` para indicar resultados erróneos en operaciones:

- `Double.POSITIVE_INFINITY`: para representar el valor ∞
- `Double.NEGATIVE_INFINITY`: para representar el valor -∞
- `Double.isNaN(valor)` para comprobar si el valor no tiene formato numérico

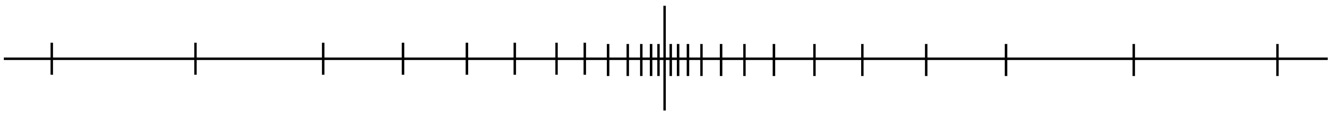
Ejemplo:

```
double a = 123, b = 0, c;  // Tres variables reales

c = a / b;  // División por cero

if (c == Double.POSITIVE_INFINITY)
    System.out.println("Error: división por cero");

double x = Math.sqrt(-1);  // Raíz cuadrada de -1

if (Double.isNaN(x))
    System.out.println("Error: resultado complejo");
```

Los números reales son un valor aproximado

Con 32 o 64 bits es imposible representar los infinitos valores existentes en la recta real

En su lugar, se representan valores aislados en una secuencia en la que están más próximos cuanto menor es su valor absoluto



Consecuencia: se representan habitualmente valores aproximados y que no son exactos, por lo que no es conveniente realizar comprobaciones de igualdad entre números reales

Ejemplo: `1.3 * 0.7` es diferente de `0.91`

Valor	IEEE 754 32 bits	Valor aproximado
1.3	00111111101001100110011001100110	1.2999999523162842
0.7	00111111001100110011001100110011	0.699999988079071
1.3 * 0.7	00111111011010001111010111000010	0.9099999666213989
0.91	00111111011010001111010111000011	0.9100000262260437

En la siguiente sentencia `if` la condición es falsa, aunque no debería serlo

```
float x, y, z;  // Declara las variables x, y, z como números reales de simple precisión

x = 1.3f;  // Asigna el valor 1.3 a la variable x
y = 0.7f;  // Asigna el valor 0.7 a la variable y
z = 0.91f;  // Asigna el valor 0.91 a la variable z

if (z == x * y) ...   // Si z es igual a x * y entonces ...
```

6.3 Conversiones de formatos numéricos

- Automáticas cuando se amplía el formato:

```
int x = 123;
float a = x;
double b = a + x;
```

- Casting necesario cuando se reduce de formato:

```
double b = 123;
float a = (float) b;
short s = (short)(a - 1);
```

6.4 Buleanos

Tipo	Tamaño	Rango
`boolean`	8 bits	true o false

- Ejemplo: `boolean b = true;`

- No es compatible con ningún formato numérico.

6.5 Caracteres

Tipo	Tamaño	Rango
`char`	16 bits	Codificación Unicode UTF-16

Estándar establecido por el *Unicode Technical Committee del Unicode Consortium*

En la versión actual se representan más de 100.000 caracteres diferentes de varios idiomas latinos, árabe, braille, cirílico, griego, etc. además de símbolos matemáticos, técnicos, musicales, etc.

Español	**Ángel, sírveme una caña**
Chino	香港「文匯報」引述高波說
Vietnamita	**Hãng được coi là ít có vấn đề này nhất**
Japonés	チバテレビカラオケ大賞
Cirílico	Правительство Греции опровергло
Griego	Προς Εργαζομένους στην Τρίτη
Árabe	**نجح فريق فالنسيا في الفوز علي نظيره**
Checo	**Ahoj, jak se máš dnes ráno v devět?**
Armenio	**Ինչպես եք այսօր առավոտյան ժամը իննին.**
Hindi	आप कैसे नौ बजे सुबह में आज कर रहे हैं
Sueco	**Saxon lågt över nordiska språken under tiden för Hansan**
etc	

En Unicode se utilizan varios formatos posibles UTF (*Unicode Transformation Format*): UTF-8, UTF-16, UTF-32

En Java se utiliza UTF-16 donde existe una tabla multilingüe BMP (*Basic Multilingual Plane*) para los códigos '\u0000' al '\uFFFF' en el que cada carácter se representa mediante un número entero de 16 bits.

Los caracteres por encima del código '\uFFFF' se representan con dos números enteros de 16 bits.

Ejemplo: `char c = 'ñ';`

Se pueden indicar por su código

```
char c = '\u5B9D';
```

宝

Caracteres especiales:

`'\r'`	Retorno de carro
`'\n'`	Alimentación de línea
`'\t'`	Tabulador
`'\''`	Comilla simple
`'\"'`	Comilla doble
`'\\'`	Barra invertida

Se utilizan en cadenas de caracteres. Ejemplos:

Cadena	Resultado
`"primera línea\nsegunda línea"`	`primera línea` `segunda línea`

`"3\t7\n438\t15\n23\t149"`	`3  7`
	`438  15`
	`23  149`

`"c:\\directorio\\fichero.txt"`	`c:\directorio\fichero.txt`
`"Tareas \"obligatorias\""`	`Tareas "obligatorias"`

6.6 Enumerados

Son variables que pueden almacenar un valor dentro de un conjunto valores identificados mediante un nombre

Ejemplo:

```
// Define un nuevo tipo de dato para representar a un día dentro de la semana
enum DiaSemana {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO}


// Declara una variable que corresponde a ese tipo de enumerado
DiaSemana hoy;

// Le asigna un valor
hoy = DiaSemana.SABADO;

if (hoy == DiaSemana.SABADO || hoy == DiaSemana.DOMINGO)
    System.out.println("En fin de semana");
else System.out.println("En laborable");
```

7 Operadores

7.1 Operadores aritméticos

- Suma `dato1 + dato2`
- Resta `dato1 - dato2`
- Multiplicación `dato1 * dato2`
- División `dato1 / dato2`
- Resto `dato1 % dato2`
- Cambio de signo `- dato`

División de números reales

```
double i = 11.0 / 3.0;  // División real, resultado = 3.6666666666
```

División de números enteros

```
int i = 11 / 3;  // División entera, resultado = 3
int j = 11 % 3;  // Resto de la división entera, resultado = 2
```

Incrementos/decrementos previos

Instrucciones	Equivalente a	Resultado

`int i = 3;`
`int j = ++i + 2;`
	`int i = 3;`
	`i = i + 1;`
	`int j = i + 2;`
		`i = 4`
		`j = 6`

`int i = 3;`
`int j = --i + 2;`
	`int i = 3;`
	`i = i - 1;`
	`int j = i + 2;`
		`i = 2`
		`j = 4`

Incrementos/decrementos posteriores

Instrucciones	Equivalente a	Resultado

`int i = 3;`
`int j = i++ + 2;`
	`int i = 3;`
	`int j = i + 2;`
	`i = i + 1;`
		`i = 4`
		`j = 5`

`int i = 3;`
`int j = i-- + 2;`
	`int i = 3;`
	`int j = i + 2;`
	`i = i - 1;`
		`i = 2`
		`j = 5`

7.2 Operadores lógicos

Comparaciones entre datos de cualquier tipo: generan buleanos

Operador	Resultado
`A < B`	Cierto si `A` menor que `B`
`A <= B`	Cierto si `A` menor o igual a `B`
`A > B`	Cierto si `A` mayor que `B`
`A >= B`	Cierto si `A` mayor o igual a `B`
`A == B`	Cierto si `A` igual a `B`
`A != B`	Cierto si `A` diferente de `B`

Operaciones entre buleanos

Operador	Función	Resultado
`A && B`	AND	Si `A` es falso, entonces el resultado es falso y no se evalúa `B`. Si A es verdadero, entonces si `B` también es verdadero, el resultado es verdadero
`A & B`	AND	Se evalúan `A` y `B`. Si ambos son verdaderos, entonces el resultado es verdadero
`A || B`	OR	Si A es verdadero, entonces no se evalúa B y el resultado es verdadero. Si A es falso, entonces se evalúa B y si B es verdadero, entonces el resultado es verdadero
`A | B`	OR	Se evalúan A y B. Si uno de los dos es verdadero, entonces el resultado es verdadero
`! A`	NOT	Negación de A
`A ^ B`	XOR	Se evalúan A y B. Resultado verdadero cuando A y B son diferentes

Ejemplo:

```
int x=3, y=5;
boolean ampliar = y/x > 1;  // Error: puede generar problemas si x == 0
boolean ampliar = (x != 0) && (y/x > 1);  // Mejor: no se evalúa y/x si x == 0
```

7.3 Operadores con asignaciones

Se pueden combinar operadores con asignación de valores cuando uno de los operandos y el resultado son la misma variable

Ejemplos

```
int x = 1, y = 2;
x += 3;      // Equivalente a   x = x + 3
x /= y - 5;  // Equivalente a   x = x / (y - 5)
```

8 Instrucciones

8.1 Instrucciones condicionales

Python	Java, C, C++, C#, PHP, etc.

`dato = -37`
`if dato < 0:`
`    absoluto = - dato`
`else:`
`    absoluto = dato`
	`int dato, absoluto;`
	`dato = -37;`
	`if (dato < 0)`
	`    absoluto = - dato;`
	`else absoluto = dato;`

	`int mes = 8;`
	`int numDias;`
	`switch (mes) {`
	`  case 1: case 3: case 5: case 7:`
	`  case 8: case 10: case 12:`
	`      numDias = 31;`
	`      break;`
	`  case 4: case 6: case 9: case 11:`
	`      numDias = 30;`
	`      break;`
	`  case 2:`
	`      numDias = 28;`
	`      break;`
	`  default:`
	`      numDias = 0;`
	`      break;`
	`}`

8.2 Bucles

Python	Java, C, C++, C#, PHP, etc.

`x = 0`
`for i in range(1, 6):`
`    x = x + i`
	`int i, x;`
	`x = 0;`
	`for (i = 1; i <= 5; i ++)`
	`    x = x + i;`

`x = 3`
`while x > 0:`
`    x = x - 1`
	`int x;`
	`x = 3;`
	`while (x > 0)`
	`    x = x - 1;`

`x = 3`
`x = x - 1`
`while x > 0:`
`    x = x - 1`
	`int x;`
	`x = 3;`
	`do`
	`    x = x - 1;`
	`while (x > 0);`

8.3 Bloques de instrucciones

Instrucciones encerradas entre llaves

Python	Java, C, C++, C#, PHP, etc.

`x = 7`
`y = 4`
`if x < y:`
`    minimo = x`
`    maximo = y`
`else:`
`    minimo = y`
`    maximo = x`
	`int x, y, minimo, maximo;`
	`x = 7;`
	`y = 4;`
	`if (x < y)`
	`  {`
	`    minimo = x;`
	`    maximo = y;`
	`  }`
	`else`
	`  {`
	`    minimo = y;`
	`    maximo = x;`
	`  }`

	`int x, y, minimo, maximo;`
	`x = 7;`
	`y = 4;`
	`if (x < y) {`
	`    minimo = x;`
	`    maximo = y;`
	`} else {`
	`    minimo = y;`
	`    maximo = x;`
	`}`

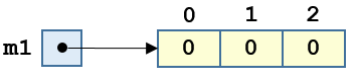
9 Matrices

Se manejan mediante referencias, como los objetos

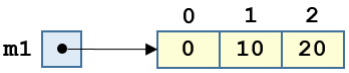
Es necesario crearlas con el operador **new**

```
int[] m1;
// Declaración de la referencia m1
```

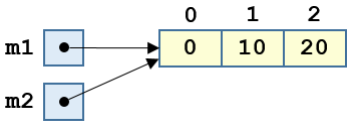

```
m1 = new int[3];
// Se crea una matriz de 3 elementos
// inicializados a 0 por defecto. La referencia
// m1 apunta a esa matriz
```



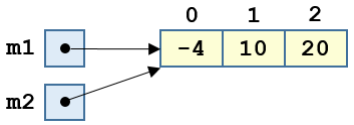

```
for (int i = 0; i < 3; i ++)
   m1[i] = i * 10;
// Modifica los valores de la matriz
```



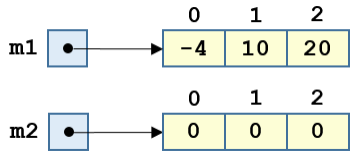

```
int[] m2;  // Crea una nueva referencia
m2 = m1;   // Las dos referencias son iguales
           // y apuntan a la misma matriz
```



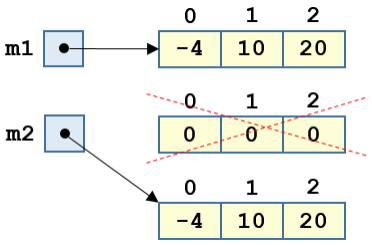

```
m1[0] = 7;
m2[0] = -4;
// Modifican la primera posición de la misma matriz
```



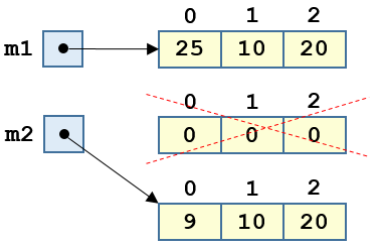

```
m2 = new int[3];
// Se crea una nueva matriz y m2 apunta a ella
```



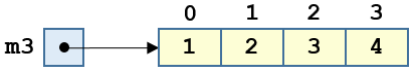

```
m2 = m1.clone();
// La matriz apuntada por m1 se clona y m2 apunta al
// clon. La matriz apuntada previamente por m2 ya no
// tiene ninguna referencia y se destruye
```




```
m1[0] = 25;
m2[0] = 9;
// Ahora se accede a matrices diferentes
```

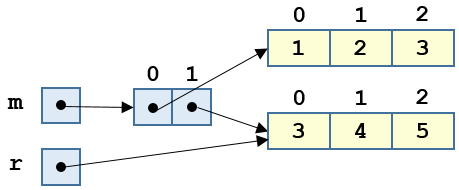



```
int[] m3 = {1, 2, 3, 4};
// Creación de una matriz a partir de unos valores
```

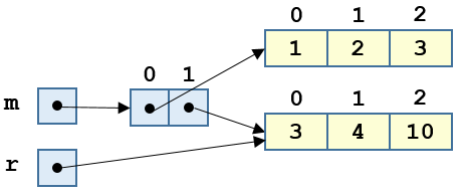


Matrices multidimensionales: en Java sólo hay matrices unidimensionales, las matrices que tienen 2 o más dimensiones se manejan con matrices adicionales de referencias

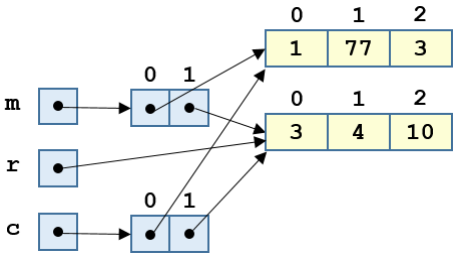
```
int[][] m = {{1, 2, 3}, {3, 4, 5}};
// Matriz bidimensional de 2 filas y 3 columnas.
// m es una referencia a una matriz de 2 referencias.

int[] r = m[1];
// Referencia a la segunda fila
```



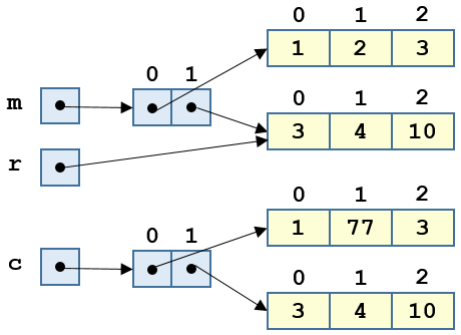

```
r[2] = 45;
m[1][2] = 10;
// Las dos instrucciones hacen lo mismo, guardan un valor
// en la segunda fila tercera columna
```




```
int[][] c = m.clone();
// Crea un clon de la matriz apuntada por m, pero
// sólo clona la matriz de las referencias

c[0][1] = 77;
// Modifica también m[0][1]
```




```
// En su lugar habría que hacer lo siguiente:

int[][] c = new int[m.length][];
// Crea una matriz de tantas referencias como filas

for (int i = 0; i < m.length; i++)
   c[i] = m[i].clone();
// Clona todas las submatrices de cada fila

c[0][1] = 77;
// Ahora la modificación es en nuevas matrices,
// no modifica m[0][1]
```



Bucle especial para iterar en una matriz.
Ejemplo: cálculo de la suma de todos los elementos

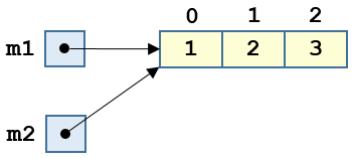
```
int[] matriz = {4, 2, 6, 7, 37, 4, 27, 9};

int suma = 0;

for (int dato : matriz)
    suma += dato;
```

Operaciones habituales con matrices implantadas en más de 100 métodos en la clase `Arrays`

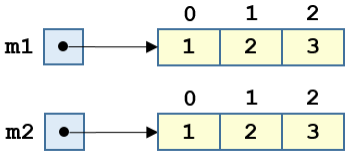
```
int[] valores = {7, 9, 2, -4, 20};  // Matriz de cinco elementos inicializados
int[] matriz = new int[5];  // Matriz de cinco elementos sin inicializar

Arrays.fill(matriz, -1);  // Asigna un valor inicial a todos los elementos

matriz = Arrays.copyOf(valores, valores.length);  // Crea una copia

Arrays.sort(matriz);  // Ordena los elementos de menor a mayor

int posicion = Arrays.binarySearch(matriz, 9);
// Busca un valor y obtiene su posición en la matriz. La matriz tiene que estar ordenada.
```

Comprobar si dos matrices son iguales:

```
int[] m1 = {1, 2, 3};  // Matriz con 3 enteros
int[] m2 = m1;  // m2 apunta a la misma matriz

boolean b = m1 == m2;
// Compara las referencias, resultado true

b = Arrays.equals(m1, m2);
// Compara los valores, resultado true
```




```
m2 = m1.clone();  // Crea un clon

b = m1 == m2;
// Compara las referencias, resultado false

b = Arrays.equals(m1, m2);
// Compara los valores, resultado true
```



10 Clases y objetos

Todo el código del programa se describe en métodos de clases

Las clases se instancian en objetos

En cada clase se encapsulan datos relacionados entre sí y que hay que manejar en bloque, junto con los métodos que procesan esos datos

Los objetos se crean dinámicamente durante la ejecución del programa utilizando el operador `new` y se utilizan referencias para poder manejarlos

Cada objeto maneja sus propios datos no estáticos definidos en la clase a la que pertenece

Todos los objetos de la misma clase comparten los datos que se hayan declarado como estáticos, marcados con la clase de almacenamiento `static`

Normalmente se definen varias clases para componer una aplicación

El identificador de cada clase comienza en mayúsculas. Ejemplo: `Punto`

Las clases marcadas con `public` son públicas y se pueden utilizar en cualquier parte del programa

Cada clase pública se codifica en un fichero con extensión `.java` y el nombre del fichero coincide con el identificador de la clase. Ejemplo: `Punto.java`

Hay que definir algún método estático `main` en alguna clase para establecer el programa principal

```
package es.uvigo.disa.pai;

public class Programa {

    public static void main(String[] args) {
        System.out.println("Hola mundo");
    }

}
```

Bloques delimitados con llaves

Instrucciones finalizadas con punto y coma

Sangría de texto para mostrar la estructura de la aplicación

Definición de la clase `es.uvigo.disa.pai.Programa` el paquete `es.uvigo.disa.pai` donde se codifica el programa principal

Clase marcada con `public` accesible a otras clases de otros paquetes. Las clases no marcadas con `public` sólo son accesibles desde el mismo paquete donde están definidas.

Método `main` marcado como `public`

Método `main` marcado como `static` actúa como un procedimiento convencional, no necesita de la instancia de ningún objeto para poder ejecutarlo

`void` para indicar que un método no devuelve ningún resultado

Las clases se agrupan en paquetes para evitar la repetición de identificadores. Cada paquete se guarda en disco en un directorio diferente.

Una clase puede utilizar directamente clases del mismo paquete

Una clase puede utilizar clases de otros paquetes mediante su inclusión con `import`

Acceder a una clase	`import java.util.Date;`
Acceder a todas las clases de un paquete	`import java.util.*;`

Para asignar un nombre a un paquete suele utilizarse el nombre de dominio Internet asignado a una compañía, para que no coincida con el nombre de otro paquete desarrollado por otros.

```
package es.uvigo.disa.pai.paquete;
```

Las clases del JDK (*Java Development Kit*) no aparecen en directorios y ficheros separados. Se almacenan en una biblioteca con extensión `.jar` en el directorio `jre/lib`

Una biblioteca contiene varias clases compiladas en bytecode y en un formato comprimido

Biblioteca `rt.jar` - Run Time Library

Una biblioteca puede contener clases de diferentes paquetes

11 Datos en una clase

Contienen valores almacenados en los objetos

Visibilidad:

- Por defecto son accesibles desde todas las clases del mismo paquete
- `private` accesibles sólo desde los métodos de la misma clase
- `public` accesibles desde cualquier método de cualquier clase
- `protected` accesibles desde la misma clase y desde clases derivadas creadas mediante herencia
- `static` datos compartidos por todos los objetos de una clase

Los datos marcados con `final` son constantes y no se pueden modificar con instrucciones, sólo admiten un valor inicial

Constantes `private static final double PI = 3.14;`

12 Métodos de una clase

Visibilidad

- `public` accesible desde cualquier otro método de cualquier clase
- `private` accesible sólo desde métodos de la misma clase
- `protected` relacionado con la herencia, se verá más adelante
- sin indicación de control de acceso: accesible desde las clases del mismo paquete

Métodos `static` no necesitan de la instancia de ningún objeto, actúan de forma similar a procedimientos y funciones convencionales

13 Constructores

Cada clase puede disponer de uno o varios constructores

Permiten indicar cómo se inicializan los objetos a partir de diferentes tipos de fuentes de información

Su identificador coincide con el de la clase

- Ejemplo 1:

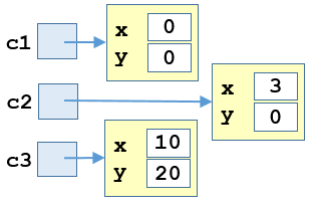
```
public class Punto {  // Un objeto representa a un punto en 2D

     public double x, y;  // Coordenadas del punto

     public Punto (double _x, double _y) {  // Constructor con dos reales
          x = _x;
          y = _y;
     }

     public Punto (double _x) { // Constructor con un real
          x = _x;
          y = 0.0;
     }

     public Punto () {  // Constructor sin parámetros
          x = 0.0;
          y = 0.0;
     }

     public Punto (Punto p) {  // Constructor de copia. Crea un objeto como copia de otro
          x = p.x;
          y = p.y;
     }
 }
```

Se puede utilizar `this` para referirse al objeto sobre el que se aplica el constructor o para llamar a otros constructores. Aplicándolo al anterior ejemplo:

```
public class Punto {  // Un objeto representa a un punto en 2D

    public double x, y;  // Coordenadas del punto

    public Punto (double x, double y) {  // Constructor con dos reales
        this.x = x;
        this.y = y;
    }

    public Punto (double x) { // Constructor con un real
        this(x, 0);  // Utiliza el primer constructor
    }

    public Punto () {  // Constructor sin parámetros
        this(0);
    }

    public Punto (Punto p) {  // Constructor de copia
        this(p.x, p.y);
    }
}
```

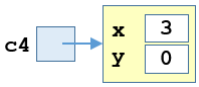
Utilización de los constructores

```
Punto c1 = new Punto();
Punto c2 = new Punto(3);
Punto c3 = new Punto(10, 20);
```



Mediante el constructor de copia se puede crear un objeto como copia de otro

```
Punto c4 = new Punto(c2);
```



14 Métodos

Los métodos pueden ser públicos o privados y pueden devolver un valor o no.

- Si no se indica nada, son accesibles a cualquier clase del mismo paquete.

- Si se marcan con `public`, son accesibles a todas las clases.

- Si se marcan con `private` son accesibles sólo dentro de su misma clase.

Ejemplo 1:

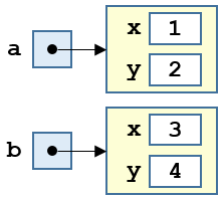
```
public class Punto {  // Un objeto representa a un punto en 2D

    public double x, y;  // Coordenadas del punto

    ...  // Constructores anteriores

    private String aCadena () {
        // Devuelve en una cadena de caracteres una representación de
        // las coordenadas del punto

        return "(" + x + ", " + y + ")";
    }
```

```
    public void visualiza () {
        // Visualiza las coordenadas del punto en la consola

        String s = aCadena();
        // Obtiene su representación en una cadena

        System.out.print(s);
        // La visualiza en pantalla
    }


    public double modulo () {  // Obtiene el módulo del vector
        return Math.sqrt(x*x + y*y);
    }
}


Punto p3 = new Punto(10,20);

p3.visualiza();
double m = p3.modulo();
```

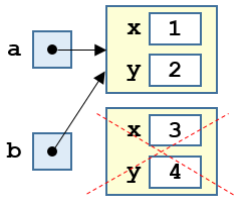
15 Objetos y referencias

Copiar referencias no implica copiar los datos de un objeto

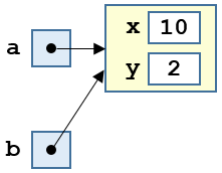
```
package es.uvigo.disa.pai;

public class Programa {

    public static void main(String[] args) {

        Punto a, b;

        a = new Punto (1, 2);  // Nuevo punto (1, 2)
        b = new Punto (3, 4);  // Nuevo punto (3, 4)

        a.visualiza();
        b.visualiza();
```




```
        b = a;  // Ambas referencias apuntan al mismo punto
                // Se destruye el objeto no referenciado, ya que
                // no se puede utilizar

        b.visualiza();  // Visualiza (1, 2)
```



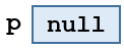

```
        a.x = 10;
        b.visualiza();  // Visualiza (10, 2)

    }
}
```



16 Referencias null

Se le puede asignar el valor `null` a una referencia, indicando así que no apunta a ningún objeto

```
Punto p = null;  // Referencia nula

...  // Resto del programa

if (p != null)  // Si p apunta a un objeto,
    ...         // procesa el objeto
```



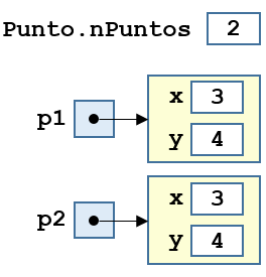
17 Datos y métodos estáticos

Los métodos estáticos permiten manejar información común a todos los objetos de una clase, pueden acceder directamente a los datos estáticos.

La utilización de datos estáticos es equivalente a la utilización de variables globales en lenguajes de programación modular. Estas variables globales se pueden repartir en una o varias clases, controlando su visibilidad. La utilización de métodos estáticos equivale al uso de subrutinas y funciones en lenguajes de programación modular.

- Ejemplo 1:

```
public class Punto {  // Un objeto representa a un punto en 2D

    public double x, y;  // Coordenadas del punto

    private static int nPuntos = 0;
    // Cuenta el número de puntos creados. Estático para que haya sólo un dato.
    // Privado para que no se pueda modificar fuera de la clase.

    public Punto (double x, double y) {  // Constructor con dos reales
        this.x = x;
        this.y = y;
        nPuntos ++;  // Se ha creado un nuevo punto
    }

    public Punto (double x) { // Constructor con un real
        this(x, 0);  // Utiliza el primer constructor
    }

    public Punto () {  // Constructor sin parámetros
        this(0, 0);
    }

    public Punto (Punto p) {  // Constructor de copia
        this(p.x, p.y);
    }

    public static int cuantosPuntos() {
        // Método estático para obtener el número de puntos creados

        return nPuntos;
    }

}
```




```
Punto p1 = new Punto(3, 4);
Punto p2 = new Punto(p1);
System.out.println("Número de puntos = " + Punto.cuantosPuntos());
```

18 Funciones matemáticas

Implantadas en la clase `java.lang.Math` mediante métodos estáticos

Constantes:
```
static double E;  // base para logaritmos neperianos
static double PI; // Número pi
```

Valor absoluto:
```
static double abs(double x)  // Devuelve el valor absoluto de x en formato double
static float abs(float x)  // Devuelve el valor absoluto de x en formato float
static int abs(int x)  // Devuelve el valor absoluto de x en formato int
```

Exponenciales:
```
static double exp(double x)  // Devuelve el número e elevado a x
static double pow(double x, double y)  // Devuelve x elevador a y
```

Logarítmicas:
```
static double log10(double x)  // Devuelve el logaritmo en base 10 de x
```
etc.

Trigonométricas:
```
static double cos(double x)  // Devuelve el coseno de x radianes
static double acos(double x)  // Devuelve el arco coseno de x en radianes
```
etc.

19 Getters y Setters

Cuando una clase expone datos públicos, es posible guardar en ellos cualquier valor dentro del rango permitido por el tipo de dato al que pertenece, lo cual puede ser un inconveniente en algunos casos.

Por ejemplo, en la siguiente clase para manejo de fechas, los datos `dia`, `mes` y `ano` son accesibles desde el programa y es posible guardar cualquier valor compatible con el tipo de datos `int`, incluso números negativos.

Ejemplo 2:

```
public class Fecha {

    // Datos públicos para guardar una fecha
    public int dia, mes, ano;

    // Constructor para inicializar una fecha sin determinar
    public Fecha() {
        mes = dia = ano = 0;
    }

    // Constructor para inicializar una fecha al día d, mes m y año a
    public Fecha(int d, int m, int a) {

        // Guarda los parámetros del constructor en el objeto
        dia = d;
        mes = m;
        ano = a;
    }

    // Un año es bisiesto si es divisible entre 4 pero no es divisible entre 100 pero
    // también es bisiesto si es divisible entre 400
    public boolean enAnoBisiesto() {
        boolean divisibleEntre4 = ano % 4 == 0;
        boolean divisibleEntre100 = ano % 100 == 0;
        boolean divisibleEntre400 = ano % 400 == 0;
        return (divisibleEntre4 && !divisibleEntre100) || divisibleEntre400;
    }
```

```
    // Obtiene el número de días que hay en el mes, teniendo en cuenta años bisiestos
    public int diasEnMes() {
       if (mes == 1 || mes == 3 || mes == 5 || mes == 7 || mes == 8 || mes == 10 || mes == 12)
            return 31;
        else if (mes == 4 || mes == 6 || mes == 9 || mes == 11)
            return 30;
        else if (enAnoBisiesto())
            return 29;
        else return 28;
    }

    // Comprueba si la fecha es válida
    public boolean valida() {
        if (mes <= 0 || mes > 12 || dia <= 0)
            return false;
        else if (dia > diasEnMes())
            return false;
        else return true;
    }

    public String aCadena() { // Devuelve una cadena con el valor de la fecha
        return dia + "/" + mes + "/" + ano;
    }

}

public class Main {

    public static void main(String[] args) {

        Fecha f = new Fecha(1, 1, 2022);  // 1 de enero de 2022

        f.dia = -79;  // Modifica el día

        System.out.println("Fecha = " + f.aCadena());
    }
}
```

Resultado: `Fecha = -79/1/2016`

Es frecuente establecer visibilidad privada para todos los datos de la clase con el objetivo de tener un mayor control sobre la información almacenada

En el caso de que sea necesario leer y/o modificar esa información, se definen los métodos de lectura/escritura correspondientes

Por convención, para el dato `x`, su método de lectura suele denominarse `getX` y su método de escritura `setX`

Ejemplo 3:

```
public class Fecha {

    private int dia, mes, ano;  // Datos privados para guardar una fecha

    private boolean valida;  // Cierto si es una fecha válida

    // Constructor para inicializar una fecha sin determinar
    public Fecha() {
        mes = dia = ano = 0;
        valida = false;
    }
```

```
// Constructor para inicializar una fecha al día d, mes m y año a
public Fecha(int d, int m, int a) {

    // Guarda los parámetros del constructor en el objeto
    dia = d;
    mes = m;
    ano = a;

    // Determina si es una fecha válida
    validaFecha();
}

// Un año es bisiesto si es divisible entre 4 pero no es divisible entre 100 pero
// también es bisiesto si es divisible entre 400
public boolean esAnoBisiesto() {
    boolean divisibleEntre4 = ano % 4 == 0;
    boolean divisibleEntre100 = ano % 100 == 0;
    boolean divisibleEntre400 = ano % 400 == 0;
    return (divisibleEntre4 && ! divisibleEntre100) || divisibleEntre400;
}

// Obtiene el número de días que hay en el mes, teniendo en cuenta años bisiestos
public int diasEnMes() {
   if (mes == 1 || mes == 3 || mes == 5 || mes == 7 || mes == 8 || mes == 10 || mes == 12)
        return 31;


   else if (mes == 4 || mes == 6 || mes == 9 || mes == 11)
        return 30;
   else if (esAnoBisiesto())
        return 29;
   else return 28;
}
// Comprueba si la fecha es válida
private void validaFecha() {
    if (mes <= 0 || mes > 12 || dia <= 0)
        valida = false;
    else if (dia > diasEnMes())
        valida = false;
    else valida = true;
}

// Ve si está validada
public boolean getValida() {
    return valida;
}

// Devuelve el día. Si la fecha no es válida, devuelve -1
public int getDia() {
    if (valida)
        return dia;
    else return -1;
}

// Modifica el día
public void setDia(int d) {
    dia = d;
    validaFecha();
}

// Devuelve el mes. Si la fecha no es válida, devuelve -1
public int getMes() {
    if (valida)
        return mes;
    else return -1;
}

// Modifica el mes
public void setMes(int m) {
    mes = m;
    validaFecha();
}
```

```
    // Devuelve el año. Si la fecha no es válida, devuelve -1
    public int getAno() {
        if (valida)
            return ano;
        else return -1;
    }

    // Modifica el año
    public void setAno(int a) {
        ano = a;
        validaFecha();
    }

    public String aCadena() { // Devuelve una cadena con el valor de la fecha
        if (valida)
            return dia + "/" + mes + "/" + ano;
        else return "inválida";
    }
}
```

En el programa:

```
Fecha f = new Fecha(1, 1, 2022);  // 1 de enero de 2022
f.setDia(-79);  // Modifica el día
System.out.println("Fecha = " + f.aCadena());
```

Resultado: `Fecha = inválida`

En cualquier caso, suele ser útil reducir las operaciones de trasvase de información en el programa, cuando sea posible. Esto permite codificar programas más simples y mejor estructurados.
Para el caso anterior, suponiendo que sólo necesitamos crear fechas, comprobar si son correctas y visualizarlas en pantalla. Ejemplo 4:

```
public class Fecha {

    private int dia, mes, ano;  // Datos privados para guardar una fecha

    private boolean valida;  // Cierto si es una fecha válida

    // Constructor para inicializar una fecha al día d, mes m y año a
    public Fecha(int d, int m, int a) {

        // Guarda los parámetros del constructor en el objeto
        dia = d;
        mes = m;
        ano = a;

        // Determina si es una fecha válida
        validaFecha();
    }

    // Un año es bisiesto si es divisible entre 4 pero no es divisible entre 100 pero
    // también es bisiesto si es divisible entre 400
    private boolean esAnoBisiesto() {
        boolean divisibleEntre4 = ano % 4 == 0;
        boolean divisibleEntre100 = ano % 100 == 0;
        boolean divisibleEntre400 = ano % 400 == 0;
        return (divisibleEntre4 && ! divisibleEntre100) || divisibleEntre400;
    }

    // Obtiene el número de días que hay en el mes, teniendo en cuenta años bisiestos
    private int diasEnMes() {
       if (mes == 1 || mes == 3 || mes == 5 || mes == 7 || mes == 8 || mes == 10 || mes == 12)
            return 31;
        else if (mes == 4 || mes == 6 || mes == 9 || mes == 11)
            return 30;
        else if (esAnoBisiesto())
            return 29;
        else return 28;
    }
```

```
    // Comprueba si es válida
    private void validaFecha() {
        if (mes <= 0 || mes > 12 || dia <= 0)
            valida = false;
        else if (dia > diasEnMes())
            valida = false;
        else valida = true;
    }

    // Ve si está validada
    public boolean getValida() {
        return valida;
    }

    // Convierte la fecha a cadena de caracteres
    public String aCadena() {
        if (valida)
            return dia + "/" + mes + "/" + ano;
        else return "inválida";
    }
}

public class Main {

    public static void main(String[] args) {
        Fecha f = new Fecha(1, 1, 2016);  // Crea un objeto fecha
        f = new Fecha(-79, 1, 2016);  // Una modificación crea otro objeto
        System.out.println("Fecha = " + f.aCadena());
        // El objeto sabe dar formato a sus datos para poder visualizarlos

    }
}
```

Resultado: `Fecha = inválida`

La definición de datos privados permite establecer en la clase un conjunto de métodos públicos para utilizar los objetos, ocultando la implantación interna de algoritmos y almacenamiento de información

Ejemplo 5: clase para manejo de números complejos

```
public class Complejo {

    public enum Inicializacion {CARTESIANA, POLAR}
    // Enumerado para poder indicar un tipo de inicialización:
    // forma cartesiana o forma polar

    // Datos para representar a un número complejo en coordenadas polares
    private double angulo;  // Ángulo en radianes con respecto al eje real
    private double modulo;   // Módulo del número complejo

    public Complejo() {  // Constructor sin parámetros
        angulo = modulo = 0;  // Inicializa a un complejo nulo
    }

    public Complejo(Inicializacion ini, double realOModulo, double imaginariaOAngulo) {
    // Constructor con los siguientes parámetros:
    // ini: tipo de inicialización
    // En el caso de una inicialización a partir de coordenadas cartesianas:
    //     realOModulo: parte real
    //     imaginariaOAngulo: parte imaginaria
    // En el caso de una inicialización a partir de coordenadas polares:
    //     realOModulo: módulo del número complejo
    //     angulo: ángulo con respecto al eje real, en radianes
```

```
        switch (ini) {  // Según el tipo de inicialización ...

            case CARTESIANA:  // En forma cartesiana
                modulo = Math.sqrt(realOModulo * realOModulo +
                        imaginariaOAngulo * imaginariaOAngulo);
                angulo = Math.atan2(imaginariaOAngulo, realOModulo);
                break;

            case POLAR:  // En forma polar
                modulo = realOModulo;
                angulo = imaginariaOAngulo;
                break;
        }
    }

    // Método público estático para conversión de ángulos de grados a radianes
    public static double gradosARadianes(double angulo) {
        return angulo * Math.PI / 180;
    }

    // Método público estático para conversión de ángulos de radianes a grados
    public static double radianesAGrados(double angulo) {
        return angulo * 180 / Math.PI;
    }

    public double getReal() {  // Devuelve la parte real del número complejo
        return modulo * Math.cos(angulo);
    }

    public double getImaginaria() {  // Devuelve la parte real del número complejo
        return modulo * Math.sin(angulo);
    }

    // Modifica la parte real del número complejo, manteniendo la misma parte imaginaria
    public void setReal(double real) {
        double imaginaria = getImaginaria();
        modulo = Math.sqrt(real * real + imaginaria * imaginaria);
        angulo = Math.atan2(imaginaria, real);
    }


    // Modifica la parte imaginaria del número complejo, manteniendo la misma parte real
    public void setImaginaria(double imaginaria) {
        double real = getReal();
        modulo = Math.sqrt(real * real + imaginaria * imaginaria);
        angulo = Math.atan2(imaginaria, real);
    }

    // Devuelve el módulo del número complejo
    public double getModulo() {
        return modulo;
    }

    // Modifica el módulo del número complejo
    public void setModulo(double m) {
        modulo = m;
    }

    // Devuelve el ángulo del número complejo
    public double getAngulo() {
        return angulo;
    }

    // Modifica el ángulo del número complejo
    public void setAngulo(double a) {
        angulo = a;
    }

    // Suma el número complejo con otro número complejo y devuelve el resultado
    public Complejo suma(Complejo otro) {
        double real = getReal() + otro.getReal();
        double imaginaria = getImaginaria() + otro.getImaginaria();
        return new Complejo(Inicializacion.CARTESIANA, real, imaginaria);
    }
```

```
    // Resta el número complejo con otro número complejo y devuelve el resultado
    public Complejo resta(Complejo otro) {
        double real = getReal() - otro.getReal();
        double imaginaria = getImaginaria() - otro.getImaginaria();
        return new Complejo(Inicializacion.CARTESIANA, real, imaginaria);
    }

    // Multiplica el número complejo por otro número complejo y devuelve el resultado
    public Complejo multiplica(Complejo otro) {
        return new Complejo(Inicializacion.POLAR, modulo * otro.modulo, angulo + otro.angulo);
    }

    // Multiplica el número complejo por un escalar y devuelve el resultado
    public Complejo multiplicaEscalar(double escalar) {
        return new Complejo(Inicializacion.POLAR, modulo * escalar, angulo);
    }

    // Divide el número complejo por otro número complejo y devuelve el resultado
    public Complejo divide(Complejo otro) {
        return new Complejo(Inicializacion.POLAR, modulo / otro.modulo, angulo - otro.angulo);
    }

    // Devuelve el número complejo conjugado
    public Complejo conjugado() {
        return new Complejo(Inicializacion.POLAR, modulo, - angulo);
    }

    // Devuelve el resultado de la suma de varios números complejos
    public static Complejo suma(Complejo ... complejos) {
        Complejo resultado = new Complejo();
        for (Complejo c : complejos)
            resultado = resultado.suma(c);
        return resultado;
    }

    // Devuelve el resultado de la resta de un número complejo (el primero) menos varios
    // números complejos (los restantes)
    public static Complejo resta(Complejo ... complejos) {
        Complejo resultado = complejos[0];
        for (int i = 1; i < complejos.length; i++)
            resultado = resultado.resta(complejos[i]);
        return resultado;
    }

    // Devuelve el resultado de la multiplicación de varios números complejos
    public static Complejo multiplica(Complejo ... complejos) {
        Complejo resultado = new Complejo(Inicializacion.POLAR, 1, 0);
        for (Complejo c : complejos)
            resultado = resultado.multiplica(c);
        return resultado;
    }

    // Devuelve el resultado de la división de un número complejo (el primero) dividido
    // entre varios números complejos (los restantes)
    public static Complejo divide(Complejo ... complejos) {
        Complejo resultado = complejos[0];
        for (int i = 1; i < complejos.length; i++)
            resultado = resultado.divide(complejos[i]);
        return resultado;
    }

}


public class Main {

    public static void main(String[] args) {

        Complejo a, b, c;
        a = new Complejo(Complejo.Inicializacion.CARTESIANA, 3, 2);  // Complejo 3+2i
        b = new Complejo(Complejo.Inicializacion.POLAR, 5, Complejo.gradosARadianes(90)); //5i
        c = a.conjugado();  // 3-2i
```

```
        c = a.divide(b);  // (3+2i)/(5i)
        c = Complejo.divide(a, b);  // (3+2i)/(5i)
        c = Complejo.resta(  // (3+2i)-(1-i)*(2+4i)
            a,
            Complejo.multiplica(
                new Complejo(Complejo.Inicializacion.CARTESIANA, 1, -1),
                new Complejo(Complejo.Inicializacion.CARTESIANA, 2, 4)
            )
        );
    }
}
```

20 Objetos inmutables

Son objetos cuyo estado (los datos almacenados en el objeto) no se puede modificar, sólo se puede inicializar en constructores

Son clases marcadas con `final` para indicar que se instancian en objetos inmutables

Todos sus datos son constantes y privados, marcados con `private final`

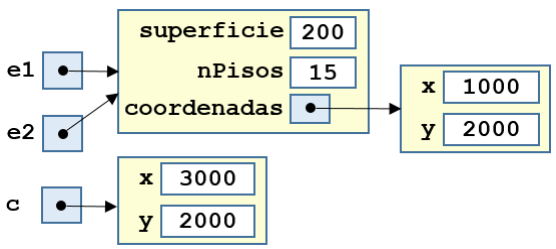
Se suelen definir los *getters* necesarios para poder leer su estado

Ventajas:

- son más fáciles de manejar, testear y construir
- no presentan problemas de sincronización cuando el programa utiliza multitarea
- no necesitan constructor de copia y tampoco es necesario clonarlos
- si surge algún problema en el programa, nunca quedan en un estado indeterminado o a medio modificar

Ejemplo 6:

```
public final class Edificio {  // Un objeto de esta clase representa a un edificio

    private final double superficie;  // Superficie en metros cuadrados
    private final int nPisos;  // Número de pisos
    private final Punto coordenadas;  // Coordenadas geográficas


    public Edificio(double _superficie, int _nPisos, double _x, double _y) { // Constructor
        superficie = _superficie;
        nPisos = _nPisos;
        coordenadas = new Punto(_x, _y);
    }


    double getSuperficie() {  // Obtiene una copia de la superficie
        return superficie;
    }

    double getNPisos() {
    // Obtiene una copia del número de pisos
        return nPisos;
    }
```



```
    Punto getCoordenadas() {  // Obtiene una copia de las coordenadas
        return new Punto(coordenadas);
    }
}


public class Main {

    public static void main(String[] args) {

        Edificio e1 = new Edificio(200, 15, 1000, 2000);
        // Edificio de 200 m2, 15 pisos en la posición (1000, 2000)

        System.out.println("Superficie de e1 = " + e1.getSuperficie());
        // Visualiza su superficie

        Punto c = e1.getCoordenadas(); // Obtiene una copia, no devuelve el original
        c.x = 3000;  // No modifica la posición del edificio

        Edificio e2 = e1;
        // No es necesario clonar objetos para representar a un edificio con las
        // mismas características

        System.out.println("Superficie de e2 = " + e2.getSuperficie());
        // Utiliza el mismo objeto
    }
}
```

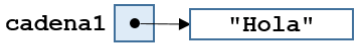
21 Cadenas de caracteres

Objetos inmutables de la clase `String`

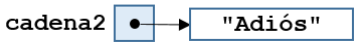
En el caso de las cadenas de caracteres, un texto indicado entre dobles comillas implica la creación de un objeto para almacenar ese texto

```
String cadena1, cadena2;  // Dos referencias

cadena1 = new String("Hola");
// Creación de un objeto de la clase String para almacenar el texto "Hola"
// La referencia cadena1 apunta a ese objeto
```

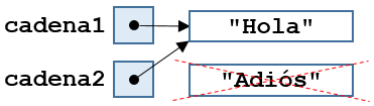
```
cadena1 = "Hola";  // Equivalente a la línea anterior
```




```
cadena2 = "Adiós";  // Otra referencia y otro objeto
```

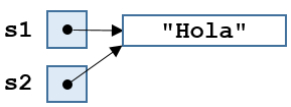



```
cadena2 = cadena1;  // Apuntan al mismo objeto
```



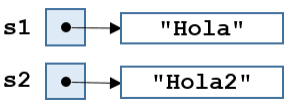
Son objetos inmutables, cualquier intento de modificación de una cadena implica la creación de otro objeto con el nuevo valor

```
String s1 = "Hola";
String s2 = "Hola";
// El compilador decide que s1 y s2 apunten al mismo objeto, no es
// necesario crear dos objetos
```



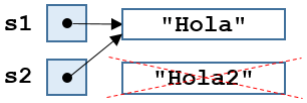

```
boolean iguales;
iguales = s1 == s2;  // Verdadero, compara las referencias y apuntan al mismo objeto
iguales = s1.equals(s2);  // Verdadero, compara el contenido y es el mismo texto
```

```
s2 = "Hola2";  // s2 apunta a otro objeto con otro valor
iguales = s1 == s2;  // Falso, apuntan a lugares diferentes
```

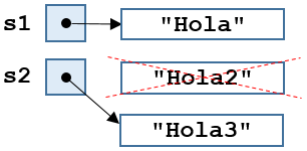


```
iguales = s1.equals(s2);  // Falso, diferente contenido
```

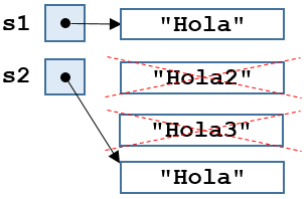
```
s2 = "Hola";  // El compilador decide que s2 vuelva a apuntar a la
              // misma cadena que apunta s1, no es neceario crear otra.
              // El segundo objeto se destruye, ya que no hay ninguna
              // referencia apuntándolo
iguales = s1 == s2;  // Cierto, apuntan al mismo sitio
iguales = s1.equals(s2);  // Verdadero, es el mismo contenido
```




```
s2 = "Hola3";  // s2 apunta a una nueva cadena
```




```
s2 = s2.substring(0, s2.length() - 1);
// Se queda con todos los caracteres desde la primera posición
// hasta el penúltimo carácter, creando otra cadena y
// destruyendo la original

iguales = s1 == s2;  // Falso, apuntan a lugares diferentes
iguales = s1.equals(s2);  // Cierto, es el mismo contenido
```



Si se desea modificar un carácter de una cadena:

```
String cadena = "Hola";
char[] caracteres = cadena.toCharArray();  // Convierte la cadena a matriz de caracteres
caracteres[0] = 'h';  // Modifica el primer carácter en la matriz
cadena = String.valueOf(caracteres);  // Crea otra cadena a partir de la matriz
```

Otras operaciones con cadenas

```
// Obtiene el número de caracteres
int longitud = cadena1.length();

// Concatenación con el operador de suma, crea un nuevo objeto con el resultado
cadena1 = cadena1 + " Pepe";

// Comprobación del contenido
boolean condicion = cadena1.startsWith("Hola");
condicion = cadena1.endsWith("Pepe");

// Búsquedas
int posicion = cadena1.indexOf("Pepe");

// Comparaciones alfabéticas
cadena1 = "Hola Pepe";
cadena2 = "Hola Ana";
int comparacion = cadena1.compareTo(cadena2);  // Resultado > 0

// Subcadenas
String nombre = cadena1.substring(5);  // Desde la posición 5 hasta el final
String inicial = cadena1.substring(5, 6);  // 1 carácter desde la posición 5

// Sustituciones
cadena1 = cadena1.replaceAll("Pepe", "Eduardo");

// Conversiones con formatos numéricos
int entero = 32;
String cadena3 = String.valueOf(entero);  // Conversión de entero a cadena
entero = Integer.parseInt(cadena3);  // Conversión de cadena a entero
cadena3 = "2.584";
double real = Double.parseDouble(cadena3);  // Conversión de cadena a double
cadena3 = String.valueOf(real);  // Conversión de double a cadena
```

```
// Conversiones con diferentes formatos enteros
entero = 32;
String cadenaBinario = Integer.toBinaryString(entero);  // A cadena en binario
entero = Integer.parseInt(cadenaBinario, 2);  // De cadena en binario a entero
String cadenaOctal = Integer.toOctalString(entero);  // A cadena en octal
entero = Integer.parseInt(cadenaOctal, 8);  // De cadena en octal a entero
String cadenaHexadecimal = Integer.toHexString(entero);  // A cadena en hexadecimal
entero = Integer.parseInt(cadenaHexadecimal, 16);  // De cadena en hexadecimal a entero

// Conversión a minúsculas o a mayúsculas
cadena1 = cadena1.toLowerCase();
cadena1 = cadena1.toUpperCase();

// Eliminación de espacios al comienzo y final
cadena1 = cadena1.trim();

// Formateo de cadenas
double x = 35.2837;
int n = 254;
cadena1 = String.format("Valor = %f", x);     // Resultado: "Valor = 35.2837"
cadena1 = String.format("Valor = %.2f", x);   // Resultado: "Valor = 35.28"
cadena1 = String.format("Valor = %d", n);     // Resultado: "Valor = 254"
cadena1 = String.format("Valor = %X", n);     // Resultado: "Valor = FE"

// División de cadenas en partes
String cadena = "36;28;495;-23";
String[] partes = cadena.split(";");
// Resultado: una matriz con 4 referencias apuntando a cadenas "36", "28", "495" y "-23"
```

En el caso de necesitar realizar muchas modificaciones en una cadena, es preferible utilizar la clase `StringBuilder`, con métodos de modificación del contenido, para evitar estar creando y destruyendo contínuamente objetos auxiliares

Ejemplo:

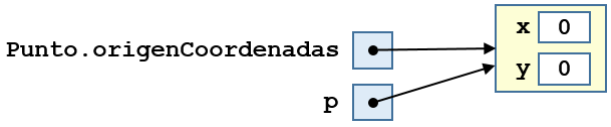
```
// Se crea un nuevo objeto en cada modificación
String cadena = "Hola";
cadena = cadena + " Gerardo";
cadena = cadena + " ¿cómo estás?";

// Todas las operaciones se realizan sobre el mismo objeto
StringBuilder sb = new StringBuilder();
sb.append("Hola");
sb.append(" Gerardo");
sb.append(" ¿cómo estás?");
String cadena = sb.toString();  // Recupera la cadena
```

22 Objetos como datos estáticos

En ocasiones es interesante añadir a una clase uno o varios objetos accesibles a partir de referencias estáticas para poder representar valores que se estima que se van a utilizar con frecuencia en un programa

Ejemplo: clase para manejo de puntos en 2D

```
public class Punto {  // Un objeto representa unas coordenadas en 2D

    public double x, y;  // Coordenadas del punto. Cada objeto tiene las suyas

    public static final Punto origenCoordenadas = new Punto();
    // Representa al origen de coordenadas (0,0). Se maneja en un objeto
    // estático y constante

    ...  // Constructores y métodos
}
```



```
// En el programa:
Punto p = Punto.origenCoordenadas;
```

Ejemplo: clase para manejo de números complejos

```
class Complejo {

    public static final Complejo i = new Complejo(Inicializacion.CARTESIANA, 0, 1);
    // Representa al número i

    // Demás miembros de la clase ...

}

// En el programa ...
Complejo c = Complejo.i;
```

23 Utilizar objetos como datos en clases

En la definición de una nueva clase es frecuente utilizar objetos de otras clases como miembros

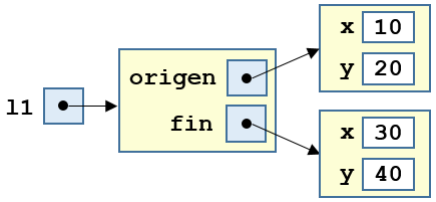
En el constructor de la nueva clase se inicializan estos objetos llamando a los constructores correspondientes

- Ejemplo 7:

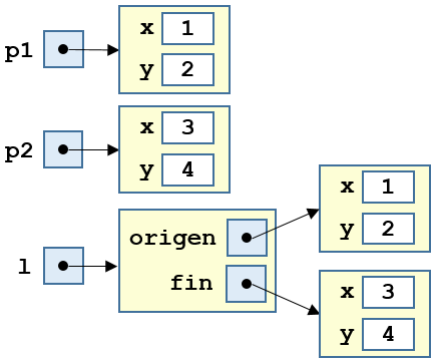
```
class Linea {

    public Punto origen, fin;
    // Un segmento de línea está representado por sus puntos origen y fin

    public Linea () {  // Constructor sin parámetros
        origen = Punto.origenCoordenadas;
        fin = Punto.origenCoordenadas;
    }

    public Linea (Punto p) {  // Constructor a partir de un punto
        origen = Punto.origenCoordenadas;
        fin = new Punto (p);
    }

    public Linea (Punto p1, Punto p2) {  // Constructor a partir de dos puntos
        origen = new Punto (p1);
        fin = new Punto (p2);
    }

    public Linea (double x1, double y1, double x2, double y2) {  // A partir de coordenadas
        origen = new Punto (x1, y1);
        fin = new Punto (x2, y2);
    }

    public Linea (Linea l) {  // Constructor de copia
        origen = new Punto (l.origen);
        fin = new Punto (l.fin);
    }

    public void visualiza () {  // Visualiza datos de la linea en la consola
        System.out.print ("Línea de ");
        origen.visualiza();
        System.out.print(" a ");
        fin.visualiza();
    }
}
```

En un programa se podría hacer lo siguiente:

```
Linea l1 = new Linea (10, 20, 30, 40);
// Línea de (10, 20) a (30, 40)

l1.visualiza();  // Visualiza las coordenadas en consola
```




```
Punto p1 = new Punto (1, 2);
Punto p2 = new Punto (3, 4);

Linea l2 = new Linea (p1, p2);
// Otra línea de (1, 2) a (3, 4)

l2.visualiza();
```



24 Herencia

Con frecuencia se definen diferentes clases con comportamientos similares. Ejemplo: los círculos y las líneas se pueden visualizar y ambos tienen una posición en el plano.

Estos comportamientos comunes y datos comunes se pueden modelar en una clase base.

Métodos abstractos: están declarados en una clase base pero sus instrucciones se definen en clases derivadas. Se marcan con `abstract`.

Clases abstractas: con métodos abstractos. Pueden tener también datos y métodos no abstractos. Se marcan con `abstract`.

Las clases derivadas reciben mediante herencia los datos y métodos de su clase base. En la definición de la clase derivada se utiliza `extends` para indicar cuál es su clase base.

Los constructores de clases derivadas llaman a constructores de la clase base para inicializar los datos recibidos mediante herencia desde la clase base.

Los miembros marcados con `protected` se pueden utilizar directamente en la misma clase y en clases derivadas.

Cuando se redefine un método en una clase derivada, se marca con `@Override` para indicar que sustituye a un método de la clase base con el mismo identificador y mismos parámetros. Esto es opcional, pero es conveniente hacerlo para que el compilador pueda realizar las comprobaciones oportunas.

Ejemplo: la clase `java.lang.Object` es por defecto clase base para todas las clases que podamos utilizar en Java.
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Object.html

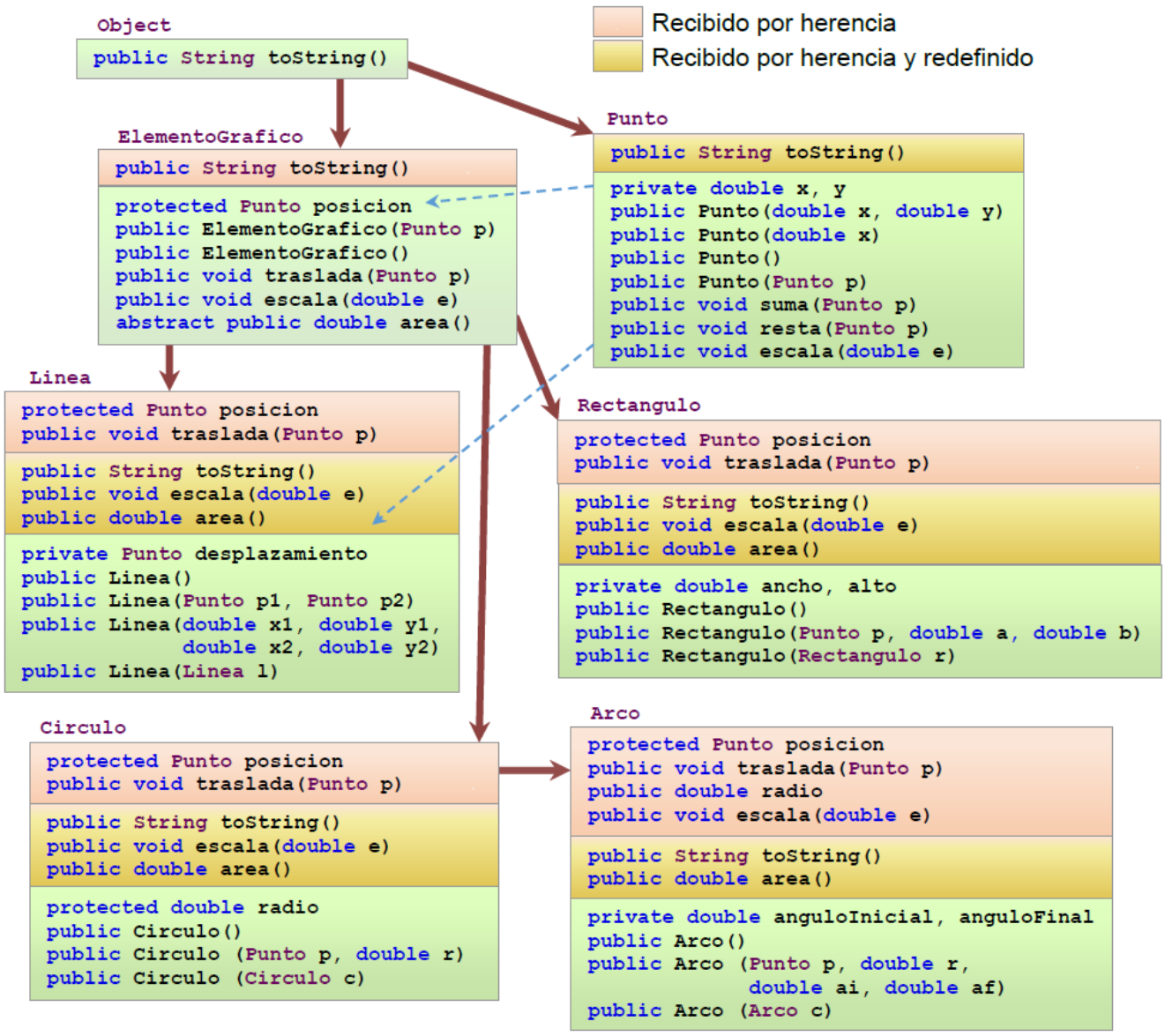
En la clase `Object` existe el método `toString` que se puede redefinir en cualquier clase:

```
class Punto {  // Un objeto representa a un punto en 2D

    public double x, y;  // Coordenadas del punto

    ... // Constructores y otros métodos
```

```
    @Override
    public String toString() {  // Redefine este método recibido por herencia

        return "Punto en (" + x + ", " + y + ")";
        // Devuelve en texto una representación del estado del objeto
    }
}


public class Main {

    public static void main(String[] args) {

        Punto p1 = new Punto(10, 20);  // Punto en (10, 20)

        String mensaje = p1.toString();  // Obtiene la representación en cadena
        System.out.println(mensaje);  // y la muestra por pantalla

        System.out.println(p1);  // Hace lo mismo
    }
}
```

Podemos utilizar la herencia para crear clases relacionadas entre sí.

Ejemplo 8:



Clase `Punto` para manejo de coordenadas 2D en el módulo `Punto.java` del paquete `es.uvigo.disa.pai`

```
package es.uvigo.disa.pai;

public class Punto {

    private double x, y;  // Coordenadas del punto


    public Punto (double _x, double _y) {  // Constructor con dos núemros reales
        x = _x;
        y = _y;
    }


    public Punto (double _x) { // Constructor con un número real
        x = _x;  // Se lo asigna a la coordenada X
        y = 0.0;  // La coordenada Y la po
    }


    public Punto () {  // Constructor sin parámetros
        x = 0.0;
        y = 0.0;
    }


    public Punto (Punto p) {  // Constructor de copia
        x = p.x;
        y = p.y;
    }


    @Override
    public String toString() {
        // Redefine este método que devuelve una representación del objeto en texto

        return "(" + x + ", " + y + ")";
    }


    public void suma (Punto p) {  // Añade las coordenadas de otro punto
        x += p.x;
        y += p.y;
    }


    public void resta (Punto p) {  // Resta las coordenadas de otro punto
        x -= p.x;
        y -= p.y;
    }


    void escala(double e) {  // Aplica un factor de escala
        x *= e;
        y *= e;
    }

}
```

Clase abstracta `ElementoGrafico` para modelar lo que es común a cualquier tipo de elemento gráfico:

```
package es.uvigo.disa.pai;

public abstract class ElementoGrafico {
    // Clase base donde hay recursos comunes para los diferentes tipos de elemetos gráficos

    protected Punto posicion;  // Posición donde está situado en el plano


    public ElementoGrafico (Punto p) {  // Constructor para situarlo en un punto
        posicion = new Punto(p);  // Crea una copia del punto y establece la posición
    }


    public ElementoGrafico () {  // Constructor para situarlo en el origen de coordenadas
        posicion = new Punto();
    }


    public void traslada (Punto t) {
    // Método de traslación para mover la posición del elemento gráfico utilizando
    // las coordenadas del punto de traslación t.
    // Este método es válido para todos los tipos de elementos gráficos

        posicion.suma(t);
    }


    public void escala(double e) {  // Escalado por el factor e
        posicion.escala(e);  // Escala la posición
    }


    public abstract double area();
    // Método abstracto, no definido en esta clase, con el objetivo de que las clases
    // derivadas lo redefinan para el cálculo del área

}
```

Clases derivadas para el manejo de diferentes elementos gráficos. Un módulo para cada una de ellas.

En `Linea.java`:

```
package es.uvigo.disa.pai;

public class Linea extends ElementoGrafico {
// Clase Linea derivada de ElementoGrafico
// Un objeto de esta clase representa a un segmento de línea entre dos puntos


    private Punto desplazamiento;
    // Desplazamiento a partir de la posición de la línea para obtener el punto final


    public Linea () {  // Constructor sin parámetros
        super();  // Inicializa la posición llamando a un constructor de la clase base
        desplazamiento = new Punto();  // Inicializa el desplazamiento con coordenadas nulas
    }
```

```
    public Linea (Punto p1, Punto p2) {  // Constructor para crear una línea que va de p1 a p2

        super(p1);  // Inicializa la posición con el primer punto

        desplazamiento = new Punto (p2);
        desplazamiento.resta(p1);
        // El desplazamiento se obtiene restando las coordenadas de los dos puntos
    }

    public Linea (double x1, double y1, double x2, double y2) {
    // Constructor partir de las coordenadas del punto inicial y final

        super(new Punto (x1, y1));  // Inicializa la posición

        desplazamiento = new Punto (x2, y2);
        desplazamiento.resta(posicion);
        // El desplazamiento se obtiene restando las coordenadas de los dos puntos
    }


    public Linea (Linea l) {  // Constructor de copia
        super (l.posicion);  // Copia la posición
        desplazamiento = new Punto(l.desplazamiento);  // Copia el desplazamiento
    }


    @Override
    public String toString () {  // Devuelve la representación de la línea en texto
        return "Línea desde " + posicion + " dirección " + desplazamiento;
    }


    @Override
    public void escala(double e) {  // Redefine el método de escalado
        super.escala(e);  // Escala la posición
        desplazamiento.escala(e);  // Escala el desplazamiento
    }


    @Override
    public double area() {  // Redefine el cálculo del área
        return 0;
    }

}
```

En `Rectangulo.java` se define una clase para manejar rectángulos situados en una determinada posición y que tienen un cierto ancho y alto:

```
package es.uvigo.disa.pai;

public class Rectangulo extends ElementoGrafico {  // Rectángulo en 2D


    private double ancho, alto;  // Dimensiones horizontal y vertical


    public Rectangulo () {  // Constructor sin parámetros

        super();  // Establece posición en el origen de coordenadas
        ancho = alto = 0;  // Dimensiones nulas
    }
```

```
    public Rectangulo (Punto p, double _ancho, double _alto) {
        // Constructor para un rectángulo situado en p y con dimensiones _ancho y _alto

        super (p);  // Establece la posición del rectángulo utilizando un constructor
                    // de la clase base

        ancho = _ancho;
        alto = _alto;
    }


    public Rectangulo (Rectangulo r) {  // Constructor de copia
        super(r.posicion);
        ancho = r.ancho;
        alto = r.alto;
    }


    @Override
    public String toString() {  // Devuelve la representación en texto
        return "Rectángulo en " + posicion + ", ancho " + ancho + ", alto " + alto;
    }


    @Override
    public void escala(double e) {  // Redefine el método de escalado
        super.escala(e);  // Escala la posición
        ancho *= e;  // Escala el ancho
        alto *= e;  // Escala el alto
    }


    @Override
    public double area() {  // Redefine el cálculo del área
        return ancho * alto;
    }

}
```

En `Circulo.java` se representa a un círculo cuyo centro está en una posición y que tiene un cierto radio:

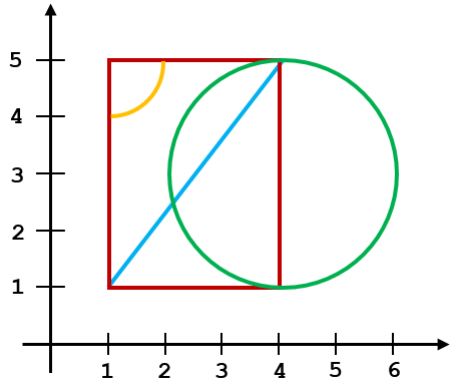
```
package es.uvigo.disa.pai;


public class Circulo extends ElementoGrafico {  // Representa a un círculo en 2D


    protected double radio;  // Radio del círculo. Protegido para utilizarlo en el arco


    public Circulo () {  // Constructor sin datos
        super();  // Posición en el origen de coordenadas
        radio = 0;  // Radio nulo
    }


    public Circulo (Punto _p, double _radio) {  // Círculo en _p, con un _radio
        super (_p);  // Llama al constructor de ElementoGrafico para inicializar la posición
        radio = _radio;  // Establece el radio
    }


    public Circulo (Circulo c) {  // Constructor de copia
        super(c.posicion);  // Copia la posición
        radio = c.radio;  // Copia el radio
    }
```

```
    @Override
    public String toString() {  // Conversión a texto
        return "Círculo en " + posicion + " y radio " + radio;
    }


    @Override
    public void escala(double e) {  // Redefine el método de escalado
        super.escala(e);  // Escala la posición
        radio *= e;  // Escala el radio
    }


    @Override
    public double area() {  // Redefine el cálculo del área
        return Math.PI * radio * radio;
    }

}
```

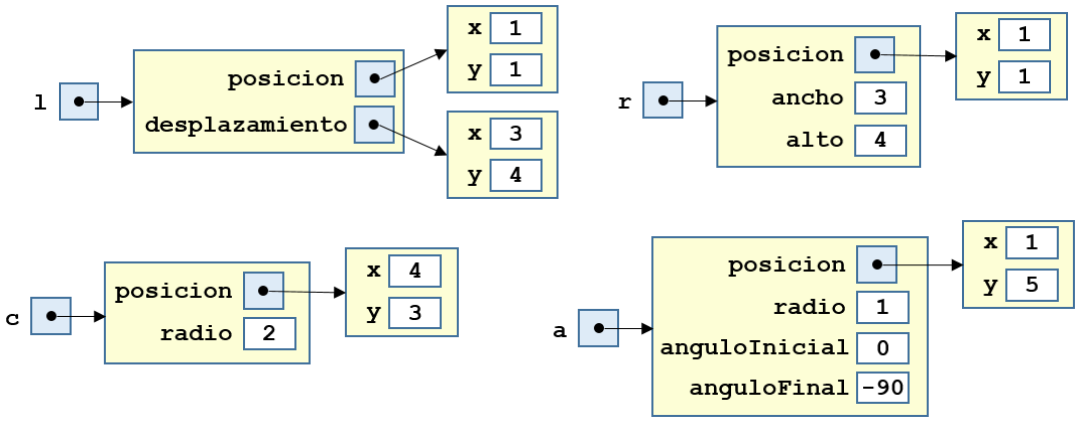
En `Arco.java` se define una clase para manejar un arco con su centro, radio, ángulo inicial y ángulo final:

```
package es.uvigo.disa.pai;


public class Arco extends Circulo {  // Representa a un arco en 2D


    private double anguloInicial, anguloFinal;  // Ángulos en grados


    public Arco () {  // Constructor sin datos
        super();  // Llama al constructor de Circulo
        anguloInicial = anguloFinal = 0;  // Ángulos nulos
    }


    public Arco (Punto p, double _radio, double _anguloInicial, double _anguloFinal) {
    // Constructor indicando el centro, el radio y los ángulos
        super (p, _radio);  // Llama al constructor de Circulo para establecer centro y radio
        anguloInicial = _anguloInicial;  // Establece ángulo inicial
        anguloFinal = _anguloFinal;  // Establece ángulo final
    }


    public Arco (Arco a) {  // Constructor de copia
        super(a.posicion, a.radio);  // Copia el centro y el radio
        anguloInicial = a.anguloInicial;  // Copia el ángulo inicial
        anguloFinal = a.anguloFinal;  // Copia el ángulo final
    }


    @Override
    public String toString() {  // Representación en cadena
        return "Arco en " + posicion + ", radio " + radio +
            ", ángulo inicial " + anguloInicial + " y ángulo final " + anguloFinal;
    }

    @Override
    public double area() {  // Redefine el cálculo del área
        return 0;
    }

}
```

Un programa que utiliza las clases definidas anteriormente:




```
package es.uvigo.disa.pai;

public class Programa {

    public static void main (String[] args) {

        Linea l = new Linea (1, 1, 4, 5);
        Rectangulo r = new Rectangulo(new Punto(1, 1), 3, 4);
        Circulo c = new Circulo(new Punto(4, 3), 2);
        Arco a = new Arco (new Punto(1, 5), 1, 0, -90);

        System.out.println("l = " + l);
        System.out.println("r = " + r);
        System.out.println("c = " + c);
        System.out.println("a = " + a);
        // Utilizan el correspondiente toString para convertir el objeto a
        // cadena y visualizarlo

        a.traslada(new Punto(0, 3));  // Traslación 3 unidades a la derecha
        System.out.println("a trasladado = " + a);

        r.escala(2);  // Escalado por 2
        System.out.println("r escalado = " + r);
        System.out.println("Área de r = " + r.area());
        // Visualiza el área del rectángulo
    }
}
```



Resultado:

```
l = Línea desde (1.0, 1.0) dirección (3.0, 4.0)
r = Rectángulo en (1.0, 1.0), ancho 3.0, alto 4.0
c = Círculo en (4.0, 3.0) y radio 2.0
a = Arco en (1.0, 5.0), radio 1.0, ángulo inicial 0.0 y ángulo final -90.0
a trasladado = Arco en (1.0, 8.0), radio 1.0, ángulo inicial 0.0 y ángulo final -90.0
r escalado = Rectángulo en (2.0, 2.0), ancho 6.0, alto 8.0
Área de r = 48.0
```

25 Métodos genéricos

Se pueden definir algoritmos genéricos que pueden aplicarse a cualquier tipo de dato

Ejemplo:

```
public static <T> String matrizACadena(T[] matriz, String separador) {
    // Método genérico para poder convertir matrices de objetos de cualquier tipo T en
    // una cadena de caracteres

    StringBuilder resultado = new StringBuilder();
    // Utiliza un StringBuilder para poder acumular todo el texto generado

    for (int i = 0; i < matriz.length; i++) {  // Recorriendo toda la matriz ...

        T elemento = matriz[i];  // Obtiene el elemento i-ésimo

        resultado.append(elemento.toString());  // Lo añade al resultado

        if (i < matriz.length-1)
            resultado.append(separador);
        // Si no es el último elemento, añade el separador
    }

    return resultado.toString();  // Devuelve el resultado como un String
}



public static void main(String[] args) {

    String[] matrizCadenas = {"A Coruña", "Lugo", "Ourense", "Pontevedra"};
    // Crea una matriz de cadenas de caracteres

    Integer[] matrizEnteros = {1, 2, 3, 4};
    // Crea una matriz de números enteros. La clase Integer permite manejarlos como objetos

    System.out.println(matrizACadena(matrizCadenas, " - "));
    System.out.println(matrizACadena(matrizEnteros, "."));
    // Convierte las matrices en cadenas de caracteres y las visualiza
}
```

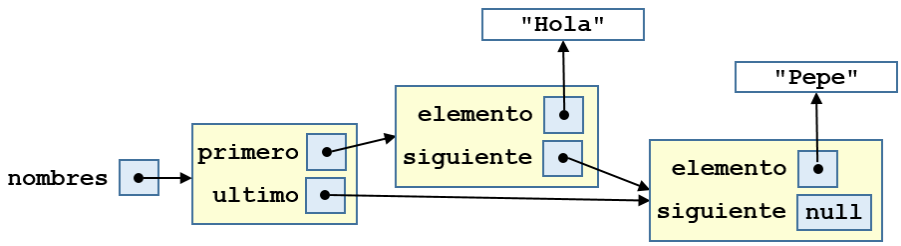
26 Clases genéricas

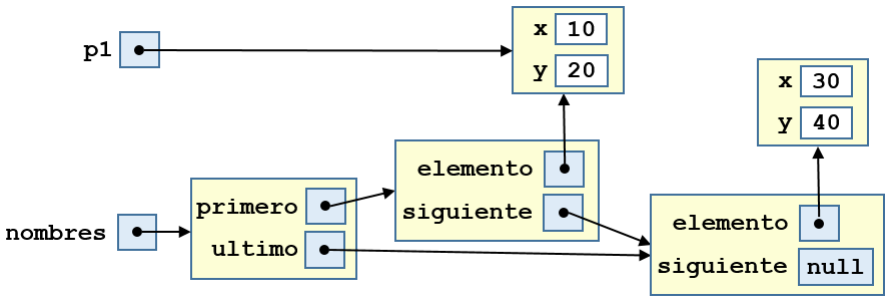
Se pueden definir clases con comportamientos genéricos que se pueden aplicar a elementos de diferentes tipos

Ejemplo 10: clase para manejar listas de elementos enlazados donde se pueden acumular objetos al final y donde se pueden extraer objetos al comienzo

```
class Lista<T> {
    // Clase genérica para poder manejar una lista enlazada de objetos de cualquier clase T


    class Nodo {  // Subclase para representar a un nodo de la lista
        T elemento;  // En cada nodo de la lista está el elemento almacenado
        Nodo siguiente;  // y hay una referencia al siguiente nodo de la lista
    }


    private Nodo primero, ultimo;  // Referencias al primero y al último nodo de la lista
```

```
    private int cuantos;  // Número de elementos almacenados


    public Lista() {  // Inicializa una lista vacía
        primero = ultimo = null;
        cuantos = 0;
    }


    public void carga(T elemento) {
        // Carga un nuevo elemento al final de la lista

        Nodo nuevo = new Nodo();  // Crea un nuevo nodo para el elemento
        nuevo.elemento = elemento;  // Apunta al nuevo elemento
        nuevo.siguiente = null;  // Está al final de la lista
        if (cuantos == 0) {  // Si la lista está vacía ...
            primero = nuevo;  // Es el prirmero
            ultimo = nuevo;   // y el último
        } else {  // si no ...
            ultimo.siguiente = nuevo;  // lo pone al final de la lista
            ultimo = nuevo;  // Indica que es el último de la lista
        }
        cuantos ++;  // Hay un elemento más
    }


    public T extrae() {  // Extrae un elemento del comienzo de la lista
        if (cuantos == 0)  // Si la lista está vacía ...
            return null;  // devuelve null
        else {  // si no ...
            T resultado = primero.elemento;  // Referencia al elemento a devolver
            primero = primero.siguiente;  // Indica que el primero es el siguiente
            cuantos --;  // Hay un elemento menos
            return resultado;  // Devuelve una referencia al elemento extraído
        }
    }

}


public class Main {

    public static void main(String[] args) {

        Lista<String> nombres = new Lista<String>();
        nombres.carga("Hola");
        nombres.carga("Pepe");
        System.out.println(nombres.extrae());
        System.out.println(nombres.extrae());

        Lista<Punto> puntos = new Lista<Punto>();
        Punto p1 = new Punto(10, 20);
        puntos.carga(p1);
        puntos.carga(new Punto(30, 40));
        System.out.println(puntos.extrae());
        System.out.println(puntos.extrae());

    }
}
```





27 Colecciones

Existen clases genéricas que permiten agrupar objetos, organizándolos de diferentes maneras

En las colecciones se almacenan referencas a los objetos cargados. No se cargan copias de los objetos.

Conjuntos - Sets

- `HashSet<T>`: conjunto de elementos no ordenados
- `LinkedHasSet<T>`: conjunto de elementos ordenados en una lista doblemente enlazada. Cada elemento tiene un enlace al siguiente y al anterior.
- `TreeSet<T>`: conjunto de elementos ordenados en árbol

Listas - Lists

- `ArrayList<T>`: matriz de elementos que se pueden acceder a partir de un subíndice. Se redimensiona automáticamente a medida que se añaden elementos.
- `Vector<T>`: similar a `ArrayList<T>`, pero se puede utilizar desde varios hilos simultáneamente, las operaciones están autosincronizadas.
- `Stack<T>`: los elementos se cargan y descargan según una organización LIFO - el último en entrar es el primero en salir - last input first output
- `LinkedList<T>`: los elementos se organizan en una lista enlazada

Colas - Queues

- `ArrayDeque<T>`: matriz redimensionable manejable como una cola de elementos enlazados
- `PriorityQueue<T>`: cola de elementos organizados mediante un valor de prioridad

Diccionarios o mapas - Maps

- `Hashtable<K,V>` y `HashMap<K,V>`: diccionario donde se puede traducir una clave de entrada `K` a un valor `V`.
- `LinkedHashMap<K,V>`: diccionario con sus entradas ordenadas en una lista

28 ArrayList

Permite acumular cualquier tipo de información y objetos en una lista

Ejemplo 11:

```
import java.util.*;

...

ArrayList<String> maquinas = new ArrayList<String>();  // Crea una lista de cadenas

maquinas.add("troqueladora");
maquinas.add("prensa");
maquinas.add("torno");
maquinas.add("cizalla");
// Añade cadenas a la lista

int numMaquinas = maquinas.size();  // Obtiene el número de cadenas

String maquina = maquinas.get(2);  // Obtiene una cadena en una posición

for (int i = 0; i < maquinas.size(); i ++)
    System.out.println(maquinas.get(i));  // Recorre la lista y visualiza todas las máquinas

for (String m : maquinas)  // Forma alternativa de recorrer un ArrayList
    System.out.println(m);

maquinas.set(0, "taladro");  // Modifica una cadena en una posición existente

ArrayList<String> copia = new ArrayList<String>(maquinas);  // Crea una copia del ArrayList

ArrayList<String> copia1 = (ArrayList<String>) maquinas.clone();
// Otra forma de crear una copia de un ArrayList

boolean mismosDatos = maquinas.equals(copia);  // Comprueba si contienen las mismas cadenas

boolean hayTorno = copia.contains("torno");  // Comprueba si contiene un valor

int posicionTorno = copia.indexOf("torno");  // Obtiene su posición

copia.remove(posicionTorno);  // Elimina una cadena de la lista por posición

copia.remove("cizalla");  // Elimina una cadena de la lista por valor

copia.clear();  // Borra toda la lista

boolean listaVacia = copia.isEmpty();  // Comprueba si está vacía

Collections.sort(maquinas);  // Ordena la lista

String[] cadenas =  new String[maquinas.size()];  // Crea una matriz de cadenas
maquinas.toArray(cadenas);  // y recibe una copia de las cadenas de la lista
```

29 HashMap

Permite crear un diccionario que obtiene un valor a partir de una clave

Ejemplo: creación de un diccionario para traducir el nombre de una substancia (una cadena de caracteres) en un valor numérico que indica su densidad

Ejemplo 12:

```
HashMap<String, Double> densidades = new HashMap<String, Double>();
// Crea un diccionario que lleva una cadena de caracteres en un número real

densidades.put("agua", 1.0);
densidades.put("aluminio", 2.7);
densidades.put("cobre", 8.92);
densidades.put("plomo", 11.3);
densidades.put("mercurio", 13.6);
densidades.put("aire", 0.00129);
densidades.put("oro", 19.3);
// Añade entradas al diccionario

double densidad;
if (densidades.containsKey("cobre"))
     densidad = densidades.get("cobre");
// Si existe la clave "cobre", obtiene su densidad

int cuantas = densidades.size();  // Número de entradas en el diccionario

densidades.remove("aluminio");  // Elimina la entrada con la clave "aluminio"

for (String material : densidades.keySet())  // Bucle para recorrer todas las claves
    System.out.println(material + " = " + densidades.get(material));

densidades.clear();  // Borra el diccionario
```

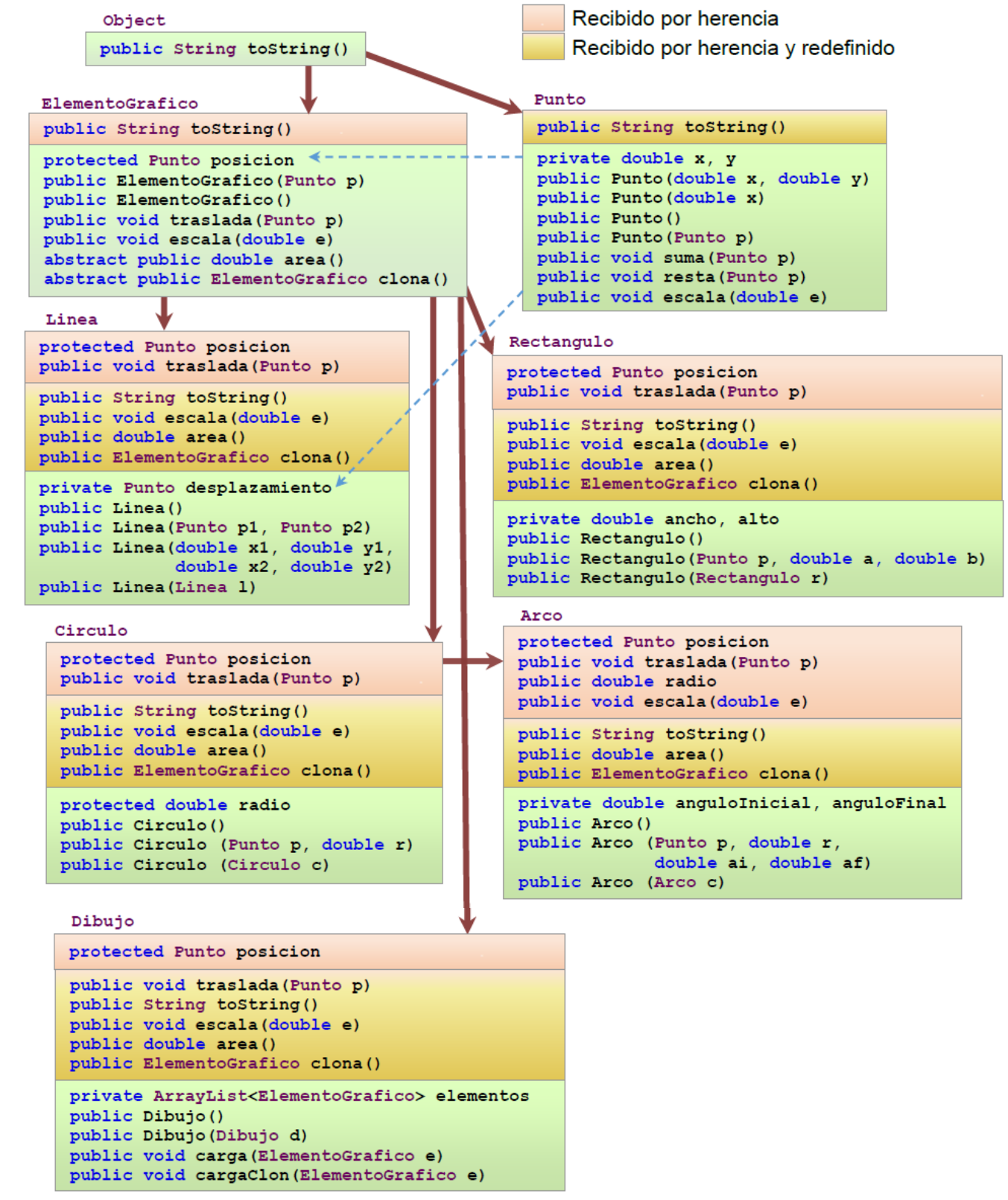
30 Polimorfismo

Existe compatibilidad entre referencias de clases derivadas y de clases base

Esto permite programar instrucciones polimórficas que pueden actuar sobre objetos de diferentes clases dentro de una misma colección (matriz, lista, diccionario, etc)

Al ejemplo analizado más atrás, añadimos la clase `Dibujo` con un `ArrayList` de elementos gráficos para manejar en bloque varios elementos de clases diferentes (líneas, rectángulos, círculos, arcos)

Se añade un comportamiento adicional a todos los elementos gráficos: operación de clonación



Ejemplo 13:

Clase base donde se añade un método abstracto de clonación:

```
public abstract class ElementoGrafico {
    // Clase base donde hay recursos comunes para los diferentes tipos de elemetos gráficos

    protected Punto posicion;  // Posición donde está situado en el plano
```

```
    public ElementoGrafico (Punto p) {  // Constructor para situarlo en un punto
        posicion = new Punto(p);  // Crea una copia del punto y establece la posición
    }


    public ElementoGrafico () {  // Constructor para situarlo en el origen de coordenadas
        posicion = new Punto();
    }


    public void traslada (Punto t) {
        // Método de traslación para mover la posición del elemento gráfico utilizando las
        // coordenadas del punto de traslación t.
        // Este método es válido para todos los tipos de elementos gráficos

        posicion.suma(t);
    }


    public void escala(double e) {  // Escalado por el factor e
        posicion.escala(e);  // Escala la posición
    }


    public abstract double area();
    // Método abstracto, no definido en esta clase, con el objetivo de que las clases
    // derivadas lo redefinan para el cálculo del área


    public abstract ElementoGrafico clona();
    // Método abstracto que tienen que redefinir las clases derivadas para crear un clon
    // del objeto sobre el cual se ejecuta, devolviendo una referencia al clon

}
```

Clases derivadas donde se implanta el método de clonación:

```
public class Linea extends ElementoGrafico {
// Clase Linea derivada de ElementoGrafico. Representa a una línea entre dos puntos.


    private Punto desplazamiento;  // Desplazamiento a partir de la posición de la línea


    public Linea () {  // Constructor sin parámetros
        super();  // Inicializa la posición llamando a un constructor de la clase base
        desplazamiento = new Punto();  // Inicializa el desplazamiento con coordenadas nulas
    }


    public Linea (Punto p1, Punto p2) {  // Constructor para crear una línea que va de p1 a p2

        super(p1);  // Inicializa la posición con el primer punto

        desplazamiento = new Punto (p2);  // El desplazamiento se obtiene
        desplazamiento.resta(p1);         // restando las coordenadas de los dos puntos
    }


    public Linea (double x1, double y1, double x2, double y2) {
        // Constructor partir de las coordenadas del punto inicial y final

        super(new Punto (x1, y1));  // Inicializa la posición

        desplazamiento = new Punto (x2, y2);  // El desplazamiento se obtiene
        desplazamiento.resta(posicion);       // restando las coordenadas de los dos puntos
    }
```

```
    public Linea (Linea l) {  // Constructor de copia
        super (l.posicion);  // Copia la posición
        desplazamiento = new Punto(l.desplazamiento);  // Copia el desplazamiento
    }


    @Override
    public String toString () {  // Devuelve la representación de la línea en texto
        return "Línea desde " + posicion + " dirección " + desplazamiento;
    }


    @Override
    public void escala(double e) {  // Redefine el método de escalado
        super.escala(e);  // Escala la posición
        desplazamiento.escala(e);  // Escala el desplazamiento
    }


    @Override
    public double area() {  // Redefine el cálculo del área
        return 0;
    }

    @Override
    public Linea clona()  {
        // Devuelve una referencia a un clon del objeto sobre el que se ejecuta. Utiliza el
        // constructor de copia para ello.

        return new Linea(this);
    }

}


public class Rectangulo extends ElementoGrafico {  // Rectángulo en 2D


    private double ancho, alto;  // Dimensiones horizontal y vertical


    public Rectangulo () {  // Constructor sin parámetros

        super();  // Establece posición en el origen de coordenadas llamando a un constructor
        // de la clase base
        ancho = alto = 0;  // Dimensiones nulas
    }


    public Rectangulo (Punto p, double _ancho, double _alto) {
        // Constructor para un rectángulo situado en p y con dimensiones _ancho y _alto

        super (p);  // Establece la posición del rectángulo utilizando un constructor
        // de la clase base

        ancho = _ancho;
        alto = _alto;
    }


    public Rectangulo (Rectangulo r) {  // Constructor de copia
        super(r.posicion);
        ancho = r.ancho;
        alto = r.alto;
    }


    @Override
    public String toString() {  // Devuelve la representación en texto
        return "Rectángulo en " + posicion + ", ancho " + ancho + ", alto " + alto;
    }
```

```
    @Override
    public void escala(double e) {  // Redefine el método de escalado
        super.escala(e);  // Escala la posición
        ancho *= e;  // Escala el ancho
        alto *= e;  // Escala el alto
    }


    @Override
    public double area() {  // Redefine el cálculo del área
        return ancho * alto;
    }


    @Override
    public Rectangulo clona() {  // Crea un clon y devuelve una referencia al mismo
        return new Rectangulo(this);
    }

}

... // La misma técnica con Circulo y Arco
```

Clase `Dibujo` con una colección de elementos gráficos y donde se utiliza polimorfismo:

```
public class Dibujo extends ElementoGrafico {
    // Un dibujo es una colección de elementos gráficos que se van a manejar en bloque. Es
    // también un elemento gráfico, es decir un dibujo puede contener a otros dibujos


    private ArrayList<ElementoGrafico> elementos;  // Un dibujo es una colección de elementos


    public Dibujo() {  // Constructor sin parámetros
        super();  // Inicializa la posición
        elementos = new ArrayList<ElementoGrafico>();  // Crea una colección vacía
    }


    public Dibujo(Dibujo d) {  // Constructor de copia
        super(d.posicion);  // Copia la posición utilizando el constructor de la clase base
        elementos = new ArrayList<ElementoGrafico>();  // Crea una nueva lista de elementos
        for (int i = 0; i < d.elementos.size(); i ++) {  // Recorriendo los elementos de d ...
            ElementoGrafico elemento = d.elementos.get(i);  // Elemento i-ésimo
            ElementoGrafico clon = elemento.clona();  // Clona el elemento
            elementos.add(clon);  // Carga el clon en el nuevo dibujo
        }
    }


    public void carga (ElementoGrafico e) {  // Añade un nuevo elemento gráfico a la colección
        elementos.add(e.clona());  // Añade un clon del elemento a cargar
    }


    @Override
    public String toString () {  // Visualiza el dibujo, redefine este método abstracto
        int i;
        StringBuilder sb = new StringBuilder();
        sb.append("Dibujo con " + elementos.size() + " elementos:\n");
        for (i = 0; i < elementos.size(); i ++) {
            sb.append("    Elemento " + i + ": ");
            sb.append(elementos.get(i).toString());
            if (i < elementos.size() - 1)
                sb.append("\n");
        }
        return sb.toString();
    }
```

```
    @Override
    public void traslada (Punto t) {
    // Redefine el método de traslación de ElementoGrafico. En este caso, hay que recorrer
    // todos los elementos cargados en el dibujo y trasladarlos uno a uno

        int i;

        for (i = 0; i < elementos.size(); i ++)  // Para cada elemento del dibujo ...
            elementos.get(i).traslada(t);  // Lo traslada
    }


    @Override
    public double area() {  // Calcula el área del dibujo
        double resultado = 0;
        for (ElementoGrafico elemento : elementos)
            resultado += elemento.area();  // Suma las áreas de sus elementos
        return resultado;
    }


    @Override
    public Dibujo clona() {  // Crea un clon de un dibujo

        return new Dibujo(this);  // Utiliza el constructor de copia para crear el clon
    }

}
```

Ejemplo de programa que utiliza estas clases:

```
public class Programa {


    public static void main(String[] args) {

        Linea l = new Linea (1, 1, 4, 5);
        Rectangulo r = new Rectangulo(new Punto(1, 1), 3, 4);
        Circulo c = new Circulo(new Punto(4, 3), 2);
        Arco a = new Arco (new Punto(1, 5), 1, 0, -90);

        Dibujo dibujo1 = new Dibujo();
        dibujo1.carga(l);
        dibujo1.carga(r);
        // Primer dibujo con la línea y el rectángulo

        Dibujo dibujo2 = new Dibujo();
        dibujo2.carga(c);
        dibujo2.carga(a);
        dibujo2.carga(dibujo1);
        // Segundo dibujo con el círculo, el arco y el primer dibujo

        dibujo2.traslada(new Punto(0, 10));
        // Traslada el segundo dibujo

        System.out.println("DIBUJO 1");
        System.out.println(dibujo1);
        System.out.println("DIBUJO 2");
        System.out.println(dibujo2);
        // Visualiza los datos de los dos dibujos
    }
}
```

Resultado:

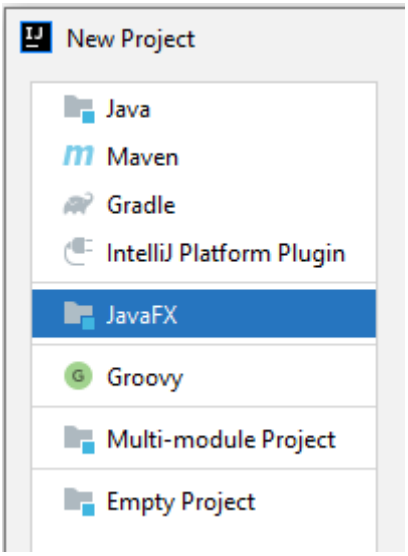
```
DIBUJO 1
Dibujo con 2 elementos:
    Elemento 0: Línea desde (1.0, 1.0) dirección (3.0, 4.0)
    Elemento 1: Rectángulo en (1.0, 1.0), ancho 3.0, alto 4.0


DIBUJO 2
Dibujo con 3 elementos:
    Elemento 0: Círculo en (4.0, 13.0) y radio 2.0
    Elemento 1: Arco en (1.0, 15.0), radio 1.0, ángulo inicial 0.0 y ángulo final -90.0
    Elemento 2: Dibujo con 2 elementos:
    Elemento 0: Línea desde (1.0, 11.0) dirección (3.0, 4.0)
    Elemento 1: Rectángulo en (1.0, 11.0), ancho 3.0, alto 4.0
```

31 JavaFX

https://openjfx.io/

Es una biblioteca de clases y herramientas para creación de interfaces gráficas en Java con soporte gráfico 2D, 3D, con animaciones, con aceleración gráfica, etc.

En Intellij Idea se crea un nuevo proyecto de tipo JavaFX



- Ejemplo 14:

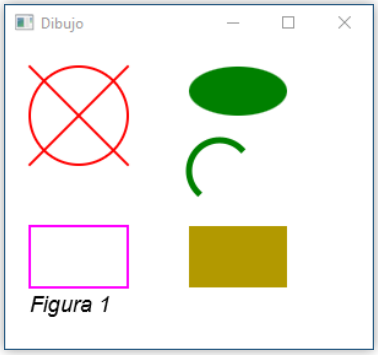


Figura 1

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.scene.shape.ArcType;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.stage.Stage;


public class Main extends Application {


    @Override
    public void start(Stage superficiePrimaria) throws Exception {
        // Este método se recibe mediante herencia y se redefine. Se ejecuta
        // al comienzo del programa. Recibe una referencia de tipo Stage para
        // acceder a la ventana donde se va a dibujar.

        superficiePrimaria.setTitle("Dibujo");  // Establece título de la ventana

        Group grupo = new Group();
        // Colección donde se van a guardar los elementos gráficos que se van a dibujar

        Canvas canvas = new Canvas(300, 250);
        // Lienzo donde se dibujan todos los elementos gráficos. Tiene unas dimensiones
        // iniciales de 300 puntos de ancho por 250 de alto. Fija las dimensiones
        // iniciales de la ventana

        GraphicsContext gc = canvas.getGraphicsContext2D();
        // Obtiene un objeto que va a permitir dibujar en el lienzo
```

```
        renderiza(gc);  // Instrucciones de dibujado

        grupo.getChildren().add(canvas);  // Asocia el canvas al grupo
        superficiePrimaria.setScene(new Scene(grupo));  // Asocia el grupo a la ventana
        superficiePrimaria.show();  // Muestra la ventana
    }


    private void renderiza(GraphicsContext gc) {
        // En este método programamos las instrucciones de dibujado de los
        // elementos gráficos utilizando los métodos de la clase
        // GraphicsContext

        gc.setStroke(Color.RED);  // Establece color de azul para trazos
        gc.setLineWidth(2);  // Establece ancho de trazo de 2 puntos
        gc.strokeLine(20, 20, 100, 100);  // Línea de (20,20) a (100,100)
        gc.strokeLine(20, 100, 100, 20);  // Línea de (20,100) a (100,20)

        gc.strokeOval(20, 20, 80, 80);
        // Circunferencia inscrita dentro de un cuadrado situado en (20,20) y de 80 de lado

        gc.setFill(Color.GREEN);  // Establece color de relleno verde
        gc.fillOval(150, 20, 80, 40);
        // Elipse situada en (150, 20), e inscrita en un rectángulo de
        // 80 de ancho y 40 de alto

        Color colorMagenta = new Color(1, 0, 1, 1);
        // Representa a un color compuesto por las componentes (valores entre 0 y 1):
        // rojo, verde, azul y alpha (transparencia)

        gc.setStroke(colorMagenta);  // Establece color de trazo
        gc.strokeRect(20, 150, 80, 50);
        // Rectángulo en (100,20) con ancho 50 y alto 30

        gc.setFill(new Color(0.7, 0.6, 0, 1));  // Establece color de relleno
        gc.fillRect(150, 150, 80, 50);
        // Rectángulo relleno de color en (150,150), ancho 80 y alto 50

        gc.setStroke(Color.GREEN);
        gc.setLineWidth(5);
        gc.strokeArc(150, 80, 50, 50, 45, 180, ArcType.OPEN);
        // Arco que es parte de un círculo inscrito en un rectángulo en (150,80)
        // de ancho 50 y alto 50, con un ángulo inicial de 45° y un ángulo
        // abarcado de 180°, dibujado con un trazo de 5 puntos

        gc.setFill(Color.BLACK);
        Font f = Font.font("Arial", FontPosture.ITALIC, 18);
        gc.setFont(f);
        gc.fillText("Figura 1", 20, 220);
        // Establece juego de caracteres Arial en itálica y 12 puntos y muestra
        // un texto en (20,220) en negro
    }


    public static void main(String[] args) {
        launch(args);
        // El programa principal contiene sólo esta instrucción, una llamada al método launch
        // recibido mediante herencia
    }

}
```

32 Interfaces

Una clase puede ser derivada de otra clase, pero sólo de una. Sin embargo, puede implantar varias interfaces.

Las interfaces son abstractas y todos sus métodos son abstractos y públicos.

Pueden contener datos, pero tienen que ser constantes, estáticos y públicos.

Existen interfaces predefinidas en las clases del sistema. Ejemplo: interfaz `Comparable` utilizada para indicar cómo se pueden comparar (menor - igual - mayor) dos objetos de una clase.

```
class Punto implements Comparable {
// Clase que implanta la interfaz Comparable para la que se define el método compareTo

    double x, y;

    ... // Mismos constructores y métodos como en casos anteriores

    @Override
    public int compareTo(Object o) {
        // Implantación del método de comparación. Se aplica sobre un objeto para
        // compararlo con el objeto que se pasa por parámetro. Hay que devolver un entero:
        //    - menor que cero si este objeto es menor que el objeto 'o'
        //    - mayor que cero si este objeto es mayor que el objeto 'o'
        //    - igual a cero si no se da ninguno de los casos anteriores

        Punto c = (Punto) o;  // Convierte a referencia de tipo Coordenadas

        double modulo1 = Math.sqrt(x * x + y * y);
        double modulo2 = Math.sqrt(c.x * c.x + c.y * c.y);
        // Obtiene el módulo del objeto sobre el que se aplica este método y el módulo del
        // objeto pasado por parámetro

        // La comparación se basa en el módulo del vector
        if (modulo1 < modulo2)
            return -1;
        else if (modulo1 > modulo2)
            return 1;
        else return 0;
    }

}
```

Gracias a la implantación de esta interfaz `Comparable` en la clase `Punto`, se puede disponer de varios objetos y ordenarlos por módulo. Ejemplo:

```
Punto[] matrizPuntos = new Punto[3];  // Matriz con referencias a 3 objetos

matrizPuntos [0] = new Punto(10, 20);
matrizPuntos [1] = new Punto (1, 2);
matrizPuntos [2] = new Punto (100, 200);
// Crea 3 objetos y los apunta desde las referencias de la matriz

Arrays.sort(matrizPuntos);  // Los ordena por módulo
```

Ejemplo 15: para los elementos gráficos se podrían definir las interfaces:

```
interface Transformable {  // Interfaz que implantan clases con transformaciones geométricas

    void traslada (Punto p);  // Traslación en el plano según las coordenadas de p

    void escala (double escala);  // Aplicación de un factor de escala

}

interface Dibujable {  // Interfaz para objetos que saben dibujarse

    void dibuja (GraphicsContext gc);
    // Método de dibujado al que se le pasa el objeto gc para la visualización en una ventana

}

interface Clonable {  // Interfaz para objetos que saben clonarse

    Object clona ();  // Método de clonado que, aplicado a un objeto, obtiene otro igual

}
```

Clase `Punto` que implanta las interfaces `Comparable`, `Transformable` y `Clonable`:

```
class Punto implements Comparable, Transformable, Clonable {
// Un punto se puede comparar, clonar y transformar, pero no dibujar

    ... // Mismos recursos como en ejemplos anteriores

    @Override
    public int compareTo(Object o) {
        // Implantación del método de comparación. Este método se aplica sobre un objeto para
        // compararlo con el objeto que se pasa por parámetro. Hay que devolver un entero:
        //    - menor que cero si este objeto es menor que el objeto 'o'
        //    - mayor que cero si este objeto es mayor que el objeto 'o'
        //    - igual a cero si no se dan los casos anteriores

        Punto c = (Punto) o;  // Convierte a referencia de tipo Coordenadas

        double modulo1 = Math.sqrt(x * x + y * y);
        double modulo2 = Math.sqrt(c.x * c.x + c.y * c.y);
        // Módulo del punto sobre el que se aplica este método y del pasado por parámetro

        // La comparación se basa en el módulo del vector correspondiente a las coordenadas
        if (modulo1 < modulo2)
            return -1;
        else if (modulo1 > modulo2)
            return 1;
        else return 0;
    }


    @Override
    public void traslada(Punto t) {  // Traslación según el vector t
        x = x + t.x;
        y = y + t.y;
    }


    @Override
    public void escala(double e) {  // Escalado con un factor e
        x = x * e;
        y = y * e;
    }


    @Override
    public Punto clona () {  // Clonación de un punto
        return new Punto(this);  // Devuelve otro punto con las mismas coordenadas
    }
}
```

Modificación de la clase base `ElementoGrafico` para que todos los elementos sean dibujables, clonables y transformables

```
abstract class ElementoGrafico implements Dibujable, Transformable, Clonable {
    // Cualquier elemento gráfico se puede dibujar, transformar y clonar


    protected Punto posicion;  // Posición donde está situado en el plano


    public ElementoGrafico (Punto p) {  // Constructor para situarlo en un punto
        posicion = new Punto(p);
    }


    public ElementoGrafico () {  // Constructor para situarlo en el origen de coordenadas
        posicion = new Punto();
    }


    @Override
    public void traslada(Punto p) {
        // La traslación se puede realizar en esta clase base, ya que aquí es donde se
        //fija la posición

        posicion.traslada(p);
    }


    @Override
    public void escala(double e) {
        // En esta clase base se escalan las coordenadas de la posición

        posicion.escala(e);
    }


    public abstract double area();
    // Método abstracto, no definido en esta clase, con el objetivo de que las clases
    // derivadas lo redefinan para el cálculo del área

}
```

Definición de los métodos de las interfaces en las clases derivadas:

```
class Linea extends ElementoGrafico {
// Clase Linea derivada de ElementoGrafico
// Un objeto de esta clase representa a un segmento de línea entre dos puntos

    private Punto desplazamiento;  // Desplazamiento a partir de la posición de la línea


    public Linea () {  // Constructor sin parámetros
        super();  // Inicializa la posición llamando a un constructor de la clase base
        desplazamiento = new Punto();  // Inicializa el desplazamiento
    }


    public Linea (Punto p1, Punto p2) {  // Constructor para crear una línea que va de p1 a p2

        super(p1);  // Inicializa la posición con el primer punto

        desplazamiento = new Punto (p2);
        desplazamiento.resta(p1);
        // El desplazamiento se obtiene restando las coordenadas de los dos puntos
    }
```

```
    public Linea (double x1, double y1, double x2, double y2) {
        // Constructor partir de las coordenadas del punto inicial y final

        super(new Punto (x1, y1));  // Inicializa la posición
        desplazamiento = new Punto (x2, y2);
        desplazamiento.resta(posicion);  // Obtiene el desplazamiento
    }


    public Linea (Linea l) {  // Constructor de copia entre dos líneas
        super (l.posicion);  // Copia la posición
        desplazamiento = new Punto(l.desplazamiento);  // Copia el desplazamiento
    }


    @Override
    public String toString () {  // Devuelve la representación de la línea en texto
        return "Línea desde " + posicion + " dirección " + desplazamiento;
    }


    @Override
    public double area() {  // Redefine el cálculo del área
        return 0;
    }


    @Override
    public void dibuja(GraphicsContext g) {
    // Dibuja la lína en una ventana con contexto gráfico g

        g.strokeLine((int) posicion.x, (int) posicion.y,
            (int) (posicion.x + desplazamiento.x), (int) (posicion.y + desplazamiento.y));
    }


    @Override
    public void escala (double e) {  // Escala las coordenadas de la línea según el factor e
        super.escala(e);  // Escala la posición utilizando el método de la clase base
        desplazamiento.escala(e);  // Escala el desplazamiento
    }


    @Override
    public Linea clona() {   // Crea un clon de la línea
        return new Linea(this);  // Utiliza el constructor de copia para crear el clon
    }
}


// Lo mismo para las clases Rectangulo, Circulo y Arco
```

Clase para representar a un dibujo como una colección de elementos gráficos.

A su vez puede actuar como otro elemento gráfico. De esta forma, puede haber dibujos que contengan otros dibujos.

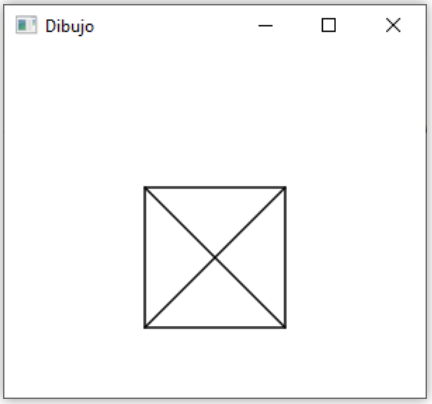
```
class Dibujo extends ElementoGrafico {  // Un dibujo es también un elemento gráfico

    ArrayList<ElementoGrafico> elementos;  // Un dibujo es una colección de elementos


    public Dibujo() {  // Constructor sin parámetros
        super();  // Inicializa la posición
        elementos = new ArrayList<ElementoGrafico>();  // Crea una colección vacía
    }
```

```
    public Dibujo(Dibujo d) {  // Constructor de copia

        super(d.posicion);  // Copia la posición utilizando el constructor de la clase base
        elementos = new ArrayList<ElementoGrafico>();  // Crea una nueva lista de elementos

        for (int i = 0; i < d.elementos.size(); i ++) {
            ElementoGrafico clon = (ElementoGrafico) d.elementos.get(i).clona();  // Clona
            elementos.add(clon);  // Carga el clon en el nuevo dibujo
        }
    }


    void carga (ElementoGrafico e) {  // Añade un nuevo elemento gráfico a la colección
        elementos.add(e);
    }


    void cargaClon (ElementoGrafico e) {  // Añade un clon de un elemento gráfico
        elementos.add((ElementoGrafico) e.clona());
    }


    @Override
    public String toString () {  // Dibujo a texto, redefine este método abstracto
        int i;
        StringBuilder sb = new StringBuilder();
        sb.append("Dibujo con " + elementos.size() + " elementos:\n");
        for (i = 0; i < elementos.size(); i ++) {
            sb.append("    Elemento " + i + ": ");
            sb.append(elementos.get(i).toString());
            if (i < elementos.size() - 1)
                sb.append("\n");
        }
        return sb.toString();
    }


    @Override
    public double area() {  // Calcula el área del dibujo
        double resultado = 0;
        for (ElementoGrafico elemento : elementos)
            resultado += elemento.area();  // Suma las áreas de sus elementos
        return resultado;
    }


    @Override
    public void dibuja (GraphicsContext g) {  // Dibuja todos los elementos en una ventana
        for(int i = 0; i < elementos.size(); i ++)    // Recorriendo todos los elementos ...
            elementos.get(i).dibuja(g);  // Los dibuja
    }


    @Override
    public void traslada (Punto p) {  // Traslada el dibujo según las coordenadas de p
        for(int i = 0; i < elementos.size(); i ++)    // Recorriendo todos los elementos ...
            elementos.get(i).traslada(p);  // Traslada cada elemento
    }


    @Override
    public void escala (double e) {  // Escala el dibujo según el factor e
        for(int i = 0; i < elementos.size(); i ++)    // Recorriendo todos los elementos ...
            elementos.get(i).escala(e);  // Escala cada elemento
    }


    @Override
    public ElementoGrafico clona() {  // Crea un clon de un dibujo
        return new Dibujo(this);  // Utiliza el constructor de copia para crear el clon
    }
}
```

Programa principal:

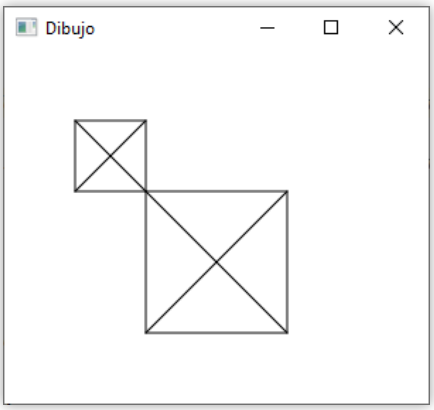
```
package es.uvigo.disa.pai;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

import java.util.ArrayList;

public class Main extends Application {


    public static void main(String[] args) {
        launch(args);  // Lanza la aplicación, provocando la ejecución del método start
    }


    @Override
    public void start(Stage superficiePrimaria) {  // Se ejecuta al comienzo del programa

        superficiePrimaria.setTitle("Dibujo");  // Establece título de la ventana
        Group grupo = new Group();  // Colección de elementos gráficos que se van a dibujar
        Canvas canvas = new Canvas(300, 250);  // Imagen donde se dibujan todos los elementos
        GraphicsContext gc = canvas.getGraphicsContext2D();  // Métodos de dibujo en canvas

        renderiza(gc);  // Instrucciones de dibujado

        grupo.getChildren().add(canvas);  // Asocia el canvas al grupo
        superficiePrimaria.setScene(new Scene(grupo));  // Asocia el grupo a la superficie
        superficiePrimaria.show();  // Muestra la superficie
    }


    private void renderiza(GraphicsContext gc) {

        Punto p1 = new Punto(50, 50);
        Punto p2 = new Punto(100, 100);
        Linea l1 = new Linea(p1, p2);
        // Crea una línea de (50,50) a (100,100) definiendo previamente dos puntos

        Linea l2 = new Linea(new Punto(50, 100), new Punto(100, 50));
        // Crea otra línea de (50,100) a (100,50)

        Rectangulo r = new Rectangulo(p1, 50, 50);
        // Crea un rectánculo en (50,50) y ancho=alto=50

        Dibujo d = new Dibujo();
        d.carga(l1);
        d.carga(l2);
        d.carga(r);
        d.escala(2);
        // Crea un dibujo y carga las líneas y el rectángulo.
        // Le aplica luego al dibujo un factor de escalado de 2.

        l1.dibuja(gc);
        l2.dibuja(gc);
        r.dibuja(gc);
        d.dibuja(gc);
        // Dibuja las líneas, el rectángulo y el dibujo

    }

}
```

Resultado si se cargan objetos sin clonar:



Si se sustituyen las operaciones de carga por cargas con clonaciones, el resultado mantiene los elementos gráficos originales y el escalado se realiza sobre los objetos clonados en el dibujo:

```
d.cargaClon(l1);
d.cargaClon(l2);
d.cargaClon(r);
```



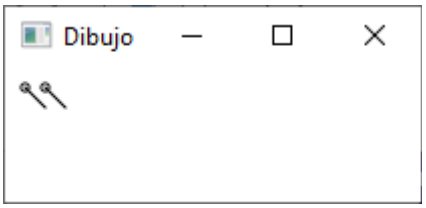
33 Clases anónimas

Cuando se implantan interfaces o se utiliza la herencia, hay que crear clases derivadas.

Si una clase derivada se va a usar puntualmente en el programa para un determinado cometido y no se va a utilizar de forma repetida para crear varios objetos, se puede definir como una clase anónima "al vuelo" justo en la instrucción donde es necesario instanciar un objeto.

Ejemplo 16: utilizar un elemento gráfico que consiste en un único punto situado en una posición.

```
Dibujo d1 = new Dibujo();  // Crea un dibujo vacío

d1.carga(new Linea(new Punto(10, 10), new Punto(20, 20)));
// Carga en d1 una línea de (10,10) a (20,20)

d1.carga(new ElementoGrafico() {

    Punto p = new Punto(10,10);

    @Override public void traslada(Punto t) {p.traslada(t);}

    @Override public void escala(double e) {p.escala(e);}

    @Override public ElementoGrafico clona() {return this;}

    @Override public void dibuja(GraphicsContext gc) {
        gc.strokeOval(p.x-2, p.y-2, 4, 4);
    }

    @Override public double area() {
        return 0;
    }
});
// Carga un nuevo elemento gráfico implantado mediante una clase anónima a partir
// de la clase ElementoGrafico, donde se redefinieron los métodos de las interfaces
// para implantar un punto que es medible, transformable, clonable y dibujable

d1.dibuja(gc);
d1.traslada(new Punto(10, 0));
d1.dibuja(gc);
// Dibuja d1 y tambien d1 trasladado
```



34 Interfaces de usuario en JavaFX

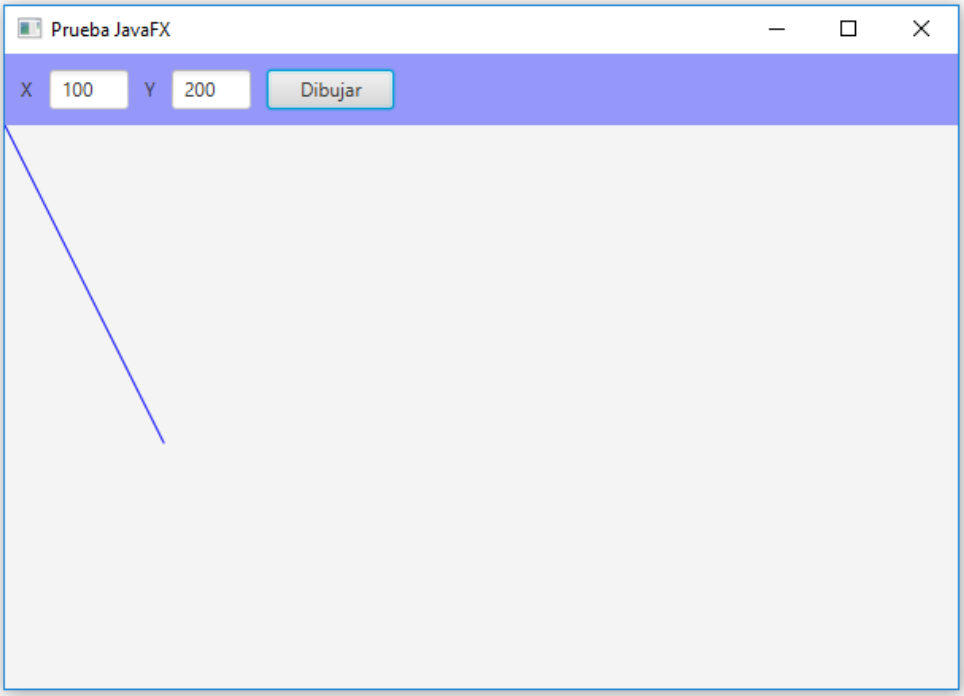
En el estándar JavaFX se pueden definir interfaces de usuario de dos formas, que no son excluyentes:

- Mediante el software Scene Builder http://gluonhq.com/products/scene-builder/ describiendo todos los componentes utilizados en una interfaz mediante un archivo XML. Este diseñador de interfaces está integrado en sistemas de desarrollo como Intellij Idea.

- Mediante código, creando los objetos necesarios para manejar todos los componentes de la interfaz. En JavaFX están normalizadas las clases de manejo de componentes de tipo botón, etiqueta, menú, cuadro de introducción de texto, imagen, etc. También existen clases que actúan como contenedores para agrupar a varios controles de una determinada forma.

Seguidamente se muestra un ejemplo del segundo caso.

Ejemplo 17:




```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage superficiePrimaria) throws Exception{

        Canvas lienzo = new Canvas();  // Crea un Canvas para dibujar en él
```

```
        lienzo.setHeight(300);  // Establece su ancho
        lienzo.setWidth(600);  // y alto
        GraphicsContext gc = lienzo.getGraphicsContext2D();  // Obtiene su contexto gráfico
        gc.setStroke(Color.BLUE);  // Establece color azul para dibujar

        Label etiquetaX = new Label("X");  // Crea una etiqueta con el texto "X"

        TextField editorX = new TextField();  // Crea un campo de edición
        editorX.setPrefSize(50, 20);  // Establece un ancho y alto

        Label etiquetaY = new Label("Y");  // Crea una etiqueta con el texto "Y"

        TextField editorY = new TextField();  // Crea un campo de edición
        editorY.setPrefSize(50, 20);  // Establece un ancho y alto

        Button boton = new Button("Dibujar");  // Crea un botón con el texto "Dibujar" encima
        boton.setPrefSize(80,20);  // Establece ancho y alto

        // Crea una clase anónima derivada de la clase EventHandler<ActionEvent> en la que
        // redefine el método handle recibido mediante herencia con el fin de indicar qué
        // código se ejecuta cuando el usuario pulsa el botón
        boton.setOnAction(new EventHandler<ActionEvent>() {

            @Override public void handle(ActionEvent e) {
                // Este método se ejecuta cuando se pulsa el botón

                int x = Integer.parseInt(editorX.getText());
                // Obtiene la coordenada X escrita en editorX

                int y = Integer.parseInt(editorY.getText());
                // Obtiene la coordenada Y escrita en editorY

                gc.strokeLine(0, 0, x, y);
                // Dibuja una línea desde (0,0) hasta esas coordenadas
            }

        });

        HBox superior = new HBox();  // Crea un panel con organización horizontal
        superior.setSpacing(10);  // Espaciado de 10 puntos entre los elementos del panel
        superior.setPadding(new Insets(10, 10, 10, 10));  // Espaciado interno de 10 puntos
        superior.setStyle("-fx-background-color: #9698F9;");  // Color de fondo en RGB
        superior.setAlignment(Pos.CENTER_LEFT);  // Los elementos alineados a la izquierda
        superior.getChildren().add(etiquetaX);  // El primer elemento es etiquetaX
        superior.getChildren().add(editorX);  // El siguiente es editorX
        superior.getChildren().add(etiquetaY);  // etc
        superior.getChildren().add(editorY);
        superior.getChildren().add(boton);

        VBox contenido = new VBox();  // Crea un panel con organización vertical
        contenido.getChildren().add(superior);  // El primer elemento es el panel superior
        contenido.getChildren().add(lienzo);  // El segundo elemento es el lienzo

        superficiePrimaria.setTitle("Prueba JavaFX");  // Establece el título de la ventana

        superficiePrimaria.setScene(new Scene(contenido, 600, 400));
        // Establece lo que se va a mostrar en la ventana y sus dimensiones iniciales

        superficiePrimaria.show();  // Muestra la ventana
    }


    public static void main(String[] args) {
        launch(args);
    }
}
```

35 Tratamiento de excepciones

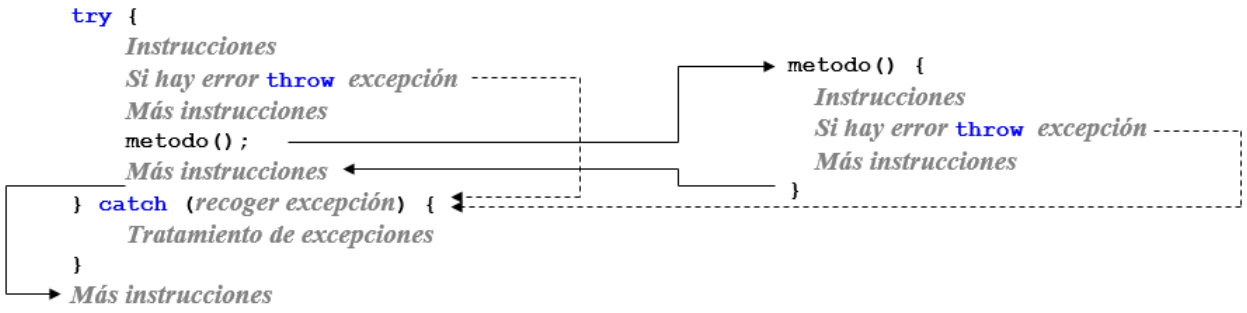
Los programas tienen que capturar y tratar los errores que se producen durante la ejecución.

Intento de ejecución de un bloque de instrucciones mediante la sentencia `try`.

Captura de errores de ejecución mediante un bloque `catch`.

Lanzamiento de excepciones mediante la sentencia `throw`.

Propagación de excepciones: si en un método se produce una excepción y no se captura, la excepción se propaga al algoritmo que llamó a ese método.



Ejemplo 18:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;


public class Main {


    static BufferedReader teclado = new BufferedReader(new InputStreamReader(System.in));
    // Para que el usuario pueda introducir un texto por teclado


    static int leeEntero() throws IOException, NumberFormatException {
        // Introduce un entero por teclado. Pueden producirse problemas de entrada de
        // información o problemas de formato, por lo que este método puede generar
        // excepciones de tipo IOException y NumberFormatException

        String cadena = teclado.readLine();  // Recoge texto hasta salto de línea
        return Integer.parseInt(cadena);  // Devuelve el texto convertido a formato numérico
    }


    static int calcula (int a, int b) throws ArithmeticException {
        // Realiza un cálculo sobre los datos pasados por parámetro.
        // Es posible que haya problemas de cálculo, como una división por cero

        return a / b;  // Devuelve el cociente de ambos datos
    }
```

```
    public static void main(String[] args) {

        int x, y, z;

        try {
            // Intenta la ejecución de las siguientes instrucciones con
            // captura de posibles excepciones

            System.out.print("Introduce un entero: ");
            x = leeEntero();
            System.out.print("Introduce otro entero: ");
            y = leeEntero();
            // Introduce dos valores por teclado

            z = calcula (x, y);
            System.out.println("Resultado = " + z);
            // Realiza un cálculo y muestra el resultado

        } catch (IOException ex) {  // Trata excepciones de entrada de información
            System.out.println("Error en teclado: " + ex.toString());

        } catch (ArithmeticException ex) {  // Trata excepciones de cálculos numéricos
            System.out.println("Error en cálculo: " + ex.toString());

        } catch (NumberFormatException ex) {  // Trata excepciones de conversión de formato
            System.out.println("Error en formato: " + ex.toString());

        }
    }

}
```

36 XML - Extended Markup Language

Es un lenguaje de descripción de información desarrollado por el W3C (*World Wide Web Consortium*).

En un archivo de texto se escribe toda la información, utilizando para ello etiquetas ó *tags* y atributos.

Las etiquetas permiten establecer una organización en árbol. Dentro de cada etiqueta se pueden establecer atributos.

Ejemplo: fichero con datos sobre varias personas, utilizando las etiquetas `datos`, `persona`, `dni`, `nombre`, `direccion`, `email`

```
<?xml version="1.0" encoding="utf-8"?>
<datos>
    <persona>
        <dni>123456</dni>
        <nombre>Ana</nombre>
        <direccion>Vigo</direccion>
        <email>ana@ejemplo.com</email>
        <email>ana@gmail.com</email>
    </persona>
    <persona>
        <dni>654321</dni>
        <nombre>Juan</nombre>
        <direccion>Monforte</direccion>
        <email>juan@ejemplo.com</email>
    </persona>
</datos>
```

Conversión de la etiqueta `dni` en un atributo

```
<?xml version="1.0" encoding="utf-8"?>
<datos>
    <persona dni="123456">
        <nombre>Ana</nombre>
        <direccion>Vigo</direccion>
        <email>ana@ejemplo.com</email>
    </persona>
    <persona dni="654321">
        <nombre>Juan</nombre>
        <direccion>Monforte</direccion>
        <email>juan@ejemplo.com</email>
    </persona>
</datos>
```

Descripción de toda la información en atributos:

```
<?xml version="1.0" encoding="utf-8"?>
<datos>
    <persona dni="123456" nombre="Ana" direccion="Vigo" email="ana@ejemplo.com" />
    <persona dni="654321" nombre="Juan" direccion="Monforte" email="juan@ejemplo.com" />
</datos>
```

En XML se utilizan espacios de nombres para diferenciar etiquetas que pueden utilizar el mismo identificador pero que tienen diferente significado. Ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<datos>
    <persona>
        <nombre>Ana</nombre>
        <edad>30</edad>
        <email>ana@ejemplo.com</email>
        <mascota>
            <especie>perro</especie>
            <nombre>Trosky</nombre>
            <edad>3</edad>
        </mascota>
    </persona>
</datos>
```

En esta información XML, las etiquetas `nombre` y `edad` tienen diferente significado según el nivel dónde se utilizan. Se pueden diferenciar claramente si pertenecen a espacios de nombres distintos

```
<?xml version="1.0" encoding="utf-8"?>
<datos xmlns:per="http://ejemplo.com/persona" xmlns:mas="http://ejemplo.com/mascota">
    <per:persona>
        <per:nombre>Ana</per:nombre>
        <per:edad>30</per:edad>
        <per:email>ana@ejemplo.com</per:email>
        <mas:mascota>
            <mas:especie>perro</mas:especie>
            <mas:nombre>Trosky</mas:nombre>
            <mas:edad>3</mas:edad>
        </mas:mascota>
    </per:persona>
</datos>
```

Cada espacio de nombres `xmlns` (*XML name space*) tiene un identificador (ejemplo: `per`) y una cadena de caracteres única (ejemplo: `="http://ejemplo.com/persona"`) que lo identifica. Esta cadena suele ser un URI (*Universal Resource Identifier*) en el que se utiliza un nombre de dominio único contratado en Internet.

Los espacios de nombres también se pueden aplicar si la información está descrita en atributos:

```
<?xml version="1.0" encoding="utf-8"?>
<datos xmlns:per="http://ejemplo.com/persona" xmlns:mas="http://ejemplo.com/mascota">
    <per:persona per:nombre="Ana" per:edad="30" per:email="ana@ejemplo.com">
        <mas:mascota mas:especie="perro" mas:nombre="Trosky" mas:edad="3" />
    </per:persona>
</datos>
```

36.1 Lectura de información XML en Java

El formato XML se utiliza habitualmente para el intercambio de información entre aplicaciones

En Java existe un conjunto de clases para poder extraer información de forma estructurada

Ejemplo: para la siguiente información guardada en el fichero `archivo.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<datos>
    <persona dni="123456">
        <nombre>Ana</nombre>
        <direccion>Vigo</direccion>
        <email>ana@ejemplo.com</email>
    </persona>
    <persona dni="654321">
        <nombre>Juan</nombre>
        <direccion>Monforte</direccion>
        <email>juan@ejemplo.com</email>
    </persona>
</datos>
```

Se puede codificar el siguiente programa para recoger datos y atributos. Ejemplo 19:

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;


public class Main {

    public static void main(String[] args) {

        // Objetos necesarios para manejar un archivo en formato XML
        Document archivoXml = null;
        DocumentBuilderFactory dbf;
        DocumentBuilder db;

        dbf = DocumentBuilderFactory.newInstance();
        // Mediante este método estático se crea un objeto de esta clase

        try {  // Intenta la ejecución de ...

            db = dbf.newDocumentBuilder();   // Crea un nuevo objeto DocumentBuilder

            archivoXml = db.parse("archivo.xml");
            // Lee toda la información escrita en archivo.xml y la interpreta

        } catch(Exception ex) {  // Si hay algún problema ...
            String causa = ex.getMessage();  // Obtiene la causa
            System.out.println("Error: " + causa);  // y la visualiza en pantalla
            System.exit(0);  // Finaliza la ejecución del programa
        }

        Element nodoRaiz = archivoXml.getDocumentElement();
        // nodoRaiz recibe una copia de toda la información leida del archivo y la
        // organiza como una colección de objetos con una estructura en forma de árbol

        NodeList listaPersonas = nodoRaiz.getElementsByTagName("persona");
        int numPersonas = listaPersonas.getLength();
        // Obtiene una colección de nodos "persona" y calcula cuántos nodos de este
        // tipo existen

        String nombre, dni;
```

```
        for (int i = 0; i < numPersonas; i ++) {
        // Recorriendo todos los nodos de tipo "persona" ...

            Element persona = (Element) listaPersonas.item(i);
            // Obtiene el nodo i-ésimo y toda la estructura en árbol que cuelga de él

            if (persona.hasAttribute("dni")) {  // Si tiene definido el atributo "dni" ...
                dni = persona.getAttribute("dni");  // obtiene su valor
            } else dni = "sin DNI";  // si no, indica que no hay DNI

            NodeList listaNombres = persona.getElementsByTagName("nombre");
            // Obtiene la lista de nodos de tipo "nombre"

            if (listaNombres != null && listaNombres.getLength() > 0) {  // Si hay alguno ...

                Element elementoNombre = (Element) listaNombres.item(0); // obtiene el primero
                nombre = elementoNombre.getFirstChild().getNodeValue();  // y su valor

            } else nombre = "sin nombre";  // si no, indica que no está definido el nombre

            System.out.println("Nombre: " + nombre + ", DNI: " + dni);
            // Visualiza la información recogida

        }

    }
}
```

En este ejemplo, si tenemos la seguridad de que cada persona tiene su nombre, se puede escribir directamente:

```
String nombre = persona.getElementsByTagName("nombre").item(0).getFirstChild().getNodeValue();
```

36.2 Escritura de información XML

Las aplicaciones pueden crear su estructura de información en memoria mediante objetos y luego volcarla en un archivo XML

Para crear el archivo XML del caso anterior. Ejemplo 20:

```
import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;


public class Main {

    public static void main(String[] args) {

        try {

            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            // Objetos necesarios para manejo de información en XML

            Document documento = db.newDocument();  // Objeto que maneja toda la información
            Element elementoRaiz = documento.createElement("datos");  // Nodo XML raíz
            documento.appendChild(elementoRaiz);  // Añade un nodo raíz "datos"
```

```
            Element persona = documento.createElement("persona");
            elementoRaiz.appendChild(persona);
            // Añade al nodo raíz un elemento de tipo "persona"

            // Le añade el atributo "dni" con el valor "123456"
            Attr atributo = documento.createAttribute("dni");
            atributo.setValue("123456");
            persona.setAttributeNode(atributo);

            // A la persona le añade un elemento "nombre" con el valor "Ana"
            Element e = documento.createElement("nombre");
            e.appendChild(documento.createTextNode("Ana"));
            persona.appendChild(e);

            // A la persona le añade un elemento "direccion" con el valor "Vigo"
            e = documento.createElement("direccion");
            e.appendChild(documento.createTextNode("Vigo"));
            persona.appendChild(e);

            // A la persona le añade un elemento "email" con el valor "ana@ejemplo.com"
            e = documento.createElement("email");
            e.appendChild(documento.createTextNode("ana@ejemplo.com"));
            persona.appendChild(e);

            // Realiza operaciones similares para otra persona y la añade al nodo raíz
            persona = documento.createElement("persona");
            elementoRaiz.appendChild(persona);
            persona.setAttribute("dni", "654321");

            e = documento.createElement("nombre");
            e.appendChild(documento.createTextNode("Juan"));
            persona.appendChild(e);

            e = documento.createElement("direccion");
            e.appendChild(documento.createTextNode("Monforte"));
            persona.appendChild(e);

            e = documento.createElement("email");
            e.appendChild(documento.createTextNode("juan@ejemplo.com"));
            persona.appendChild(e);

            // Objetos para transformar la información XML que se maneja en memoria
            // a información guardada un archivo XML en disco
            TransformerFactory tf = TransformerFactory.newInstance();
            Transformer transformer = tf.newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");

            // Guarda la información en el archivo
            DOMSource fuente = new DOMSource(documento);
            StreamResult resultado = new StreamResult(new File("archivo.xml"));
            transformer.transform(fuente, resultado);

            System.out.println("Guardado");

        // Si salta alguna excepción, se captura y se muestra un mensaje de error en pantalla

        } catch (ParserConfigurationException pce) {
            System.out.println("Error" + pce.getMessage());
            System.exit(0);
        } catch (TransformerException tfe) {
            System.out.println("Error" + tfe.getMessage());
            System.exit(0);
        }
    }
}
```

37 Expresiones Lambda

Permiten utilizar técnicas de programación funcional en Java.

Se pede crear una referencia que señala a una expresión Lambda en la que se describe un algoritmo.

Ejemplo 21:

```
// Define una interfaz funcional, donde existe un un único método que opera dos números reales
// y genera como resultado otro número real
interface OperacionMatematica {
    double operacion (double a, double b);
}


public class Main {

    // Este método recibe dos números reales y la expresión lambda que contiene
    // el algoritmo utilizado para operar los datos y generar un resultado
    public static double opera(double a, double b, OperacionMatematica o) {
        return o.operacion(a, b);
    }


    public static void main(String[] args) {

        OperacionMatematica operacionSuma = (double a, double b) -> a + b;
        // Expresión Lambda que define la suma entre dos números reales

        OperacionMatematica operacionMaximo = (double a, double b) -> {
            double resultado;
            if (a < b)
                resultado = b;
            else resultado = a;
            return resultado;
        };
        // Expresión Lambda que define un algoritmo que obtiene el máximo de
        // dos números reales

        double x = 3, y = 4, z;

        z = opera(x, y, operacionSuma);
        // Pasa la expresión Lambda operacionSuma al método opera para indicar el
        // tipo de operación a realizar entre los dos primeros parámetros, en este
        // caso, la suma

        z = opera(x, y, operacionMaximo);
        // Lo mismo para aplicar la operación que obtiene el máximo de los dos
        // valores
    }
}
```

38 Dibujo 3D en JavaFX

En JavaFX existen clases para poder trabajar en 3D:

- **`Point3D`**: puntos y vectores 3D con producto escalar y vectorial, módulo, transformaciones geométricas, etc.

- **`Node`**: elemento geométrico 3D genérico, se utiliza como clase base.

- **`Box`**, **`Sphere`**, **`Cylinder`**: paralelepípedos, esferas, cilindros. Son clases derivadas de la clase **`Node`**.

- **`Group`**: colección de objetos de clases derivadas de **`Node`**. Es a su vez otra clase derivada de **`Node`**.

- **`Transformation`**, **`Rotate`**, **`Translate`**, etc.: transformaciones geométricas.

- **`PerspectiveCamera`**: cámara con la que se puede realizar una proyección en perspectiva de una escena 3D.

- **`PointLigth`**, **`AmbientLigth`**: fuentes de luz puntuales o difusas para iluminar la escena.

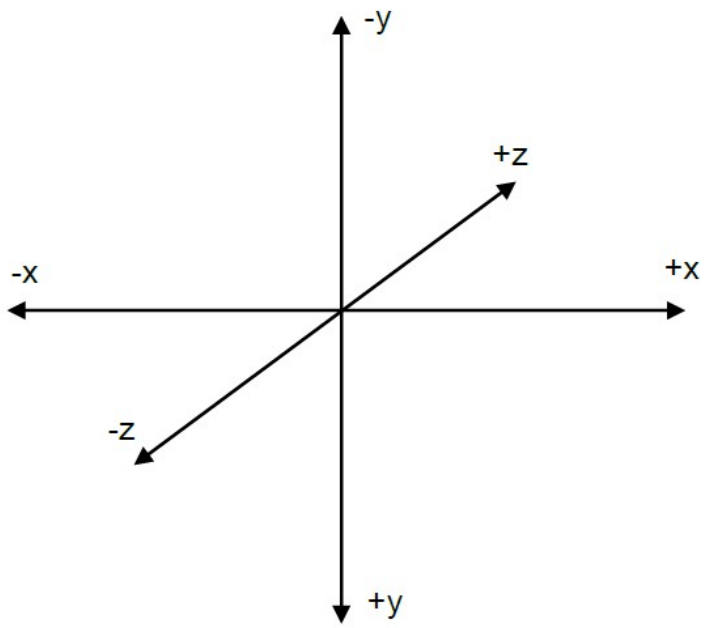
- etc.

Los objetos sólidos representados en la escena pueden corresponder a tipos de objetos predefinidos (**`Cylinder`**, **`Sphere`**, etc) o pueden ser objetos definidos por el usuario a base de unir formas triangulares, indicando sus vértices, orientación y tipo de superficie (color, textura, etc).

Las clases **`Cylinder`**, **`Sphere`**, etc son clases derivadas de la clase base **`Node`**, donde se dispone de características comunes a todo tipo de formas geométricas tridimensionales.

Toda esta información se procesa finalmente en GPUs en los tarjetas gráficas mediante técnicas de aceleración hardware, generando la proyección de las escenas, eliminando caras ocultas, tratando efectos de iluminación, etc.

Los ejes de coordenadas se establecen de la siguiente forma:



Seguidamente se muestra un ejemplo, donde se genera una escena y donde el usuario puede mover la posición de la cámara con el teclado y girarla utilizando el ratón. El usuario también puede hacer click sobre objetos de tipo **`Sphere`** para seleccionarlos. Ejemplo 22:

```
/* Escena 3D con navegación primera persona:
 - Teclas W y S para movimiento de cámara adelante/atrás.
 - Teclas A y D para movimiento de cámara izquierda/derecha.
 - Teclas E y C para movimiento de cámara arriba/abajo.
 - Con mayúsculas, el movimiento es más rápido.
 - Manteniendo pulsado el botón izquierdo del ratón y moviéndolo arriba/abajo
   e izquierda/derecha en pantalla, la cámara gira sobre sí misma en esas direcciones.
 - Rotando la rueda del ratón la cámara gira sobre su propio eje de visión.
 */

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.geometry.Insets;
import javafx.geometry.Orientation;
import javafx.geometry.Point3D;
import javafx.scene.*;
import javafx.scene.control.CheckBox;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.ToolBar;
import javafx.scene.input.KeyCode;
import javafx.scene.input.MouseEvent;
import javafx.scene.input.ScrollEvent;
import javafx.scene.layout.BorderPane;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Box;
import javafx.scene.shape.Cylinder;
import javafx.scene.shape.Sphere;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Translate;
import javafx.stage.Stage;
import java.util.ArrayList;


public class HelloApplication extends Application {

    private Group grupo3D;
    // Colección de elementos gráficos que se van a representar en 3D

    private Group grupoEjes;
    // Colección de elementos gráficos que se van a representar a los ejes de
    // coordenadas mediante paralelepípedos de la clase Box

    private PerspectiveCamera camara;
    // Cámara utilizada en la escena 3D

    private double ratonX, ratonY;
    // Posición del ratón en la ventana

    private double ratonXVentanaAntes, ratonYVentanaAntes;
    // Para recordar la posición anterior del ratón en la ventana

    public Cylinder creaLinea(Point3D origen, Point3D fin, double radioBase, Color color) {
        // Crea una línea 3D para JavaFX que une los puntos origen y fin. Consiste en un
        //  cilindro donde las caras circulares están centradas en esos puntos y tienen el
        //  radio indicado en el último parámetro.

        Point3D j = new Point3D(0, 1, 0);  // Vector unitario en el eje Y
        Point3D v = fin.subtract(origen);  // Vector que va de origen a fin
        double longitud = v.magnitude();  // Longitud de la línea

        Cylinder cilindro = new Cylinder(radioBase, longitud);
        // Crea un cilindro alineado con el eje Y, con el radio de la base radioBase y con la
        // altura indicada en longitud. Está centrado en el origen (0,0,0)

        Point3D puntoMedio = fin.midpoint(origen);  // Punto medio de la línea
```

```
        Translate traslacion = new Translate(puntoMedio.getX(), puntoMedio.getY(),
            puntoMedio.getZ());
        // Operación de traslación donde el vector de traslación tiene las coordenadas del
        // punto medio

        Point3D ejeDeRotacion = v.crossProduct(j);
        // Eje de rotación para rotar el cilindro (que está alineado con el eje Y) para
        // orientarlo en la dirección del vector v

        double anguloRotacion = Math.acos(v.normalize().dotProduct(j));
        // Ángulo en radianes que hay que aplicar al cilindro para orientarlo en la dirección
        // de v

        Rotate rotacion = new Rotate(-Math.toDegrees(anguloRotacion), ejeDeRotacion);
        // Operación de rotación un ángulo anguloRotacion (hay que indicarlo en grados)
        // alrededor de ese eje de rotación

        cilindro.getTransforms().addAll(traslacion, rotacion);
        // Aplica la traslación y la rotación al cilindro para que vaya del punto origen al
        // punto final

        cilindro.setMaterial(new PhongMaterial(color));
        // Establece el color del cilindro

        return cilindro;  // Devuelve el cilindro que representa a la línea
    }


    @Override
    public void start(Stage stage) {
        // Se ejecuta al comienzo del programa

        // -------------------- Subescena 3D ----------------------------

        Box ejeX = new Box(1000, 10, 10);
        // Crea un paralelepípedo de 1000 unidades de longitud en el eje X, 10 en el Y y 10
        // en el Z. Está centrado en el origen (0,0,0). Se utiliza para visualizar el eje X.

        ejeX.setTranslateX(500);
        // Le aplica una traslación en el eje X de 500 unidades

        ejeX.setMaterial(new PhongMaterial(Color.YELLOW));
        // Establece color amarillo

        Box ejeY = new Box(10, 1000, 10);
        ejeY.setTranslateY(500);
        ejeY.setMaterial(new PhongMaterial(Color.GREEN));
        // Box para representar al eje Y en verde

        Box ejeZ = new Box(10, 10, 1000);
        ejeZ.setTranslateZ(500);
        ejeZ.setMaterial(new PhongMaterial(Color.BLUE));
        // Box para representar al eje Z en azul

        grupoEjes = new Group();
        grupoEjes.getChildren().add(ejeX);
        grupoEjes.getChildren().add(ejeY);
        grupoEjes.getChildren().add(ejeZ);
        // Se crea una colección donde se guardan los tres Box para representar
        // a los ejes de coordenadas

        grupo3D = new Group();
        // Crea la colección de elementos gráficos que se van a visualizar en 3D

        grupo3D.getChildren().add(grupoEjes);
        // Se añade el grupo de los ejes al grupo que se va a mostrar en 3D

        Color colorRojo = new Color(1, 0, 0, 1);
        // Objeto para representar al color rojo (componentes R=1, G=0, B=0 y alpha=1).
        // Todos los valores son reales entre 0 y 1.

        Color colorMagenta = new Color(1, 0, 1, 1);
        // Color magenta, combinando rojo y azul
```

```
Sphere esfera1 = new Sphere(50);
esfera1.getTransforms().addAll(new Translate(100, 100, 100));
esfera1.setMaterial(new PhongMaterial(colorRojo));
grupo3D.getChildren().add(esfera1);
// Añade una esfera de color rojo trasladada a (100,100,100) y radio 50

Sphere esfera2 = new Sphere(50);
esfera2.getTransforms().addAll(new Translate(200, 200, 200));
esfera2.setMaterial(new PhongMaterial(colorMagenta));
grupo3D.getChildren().add(esfera2);
// Añade una esfera de color magenta en (200,200,200) y radio 50

Cylinder linea = creaLinea(new Point3D(100, 100, 100), new Point3D(200, 200, 200),
    10, new Color(0.3, 1, 0.3, 1));
// Crea una línea que une los puntos (100, 100, 100) y (200, 200, 200) con un
// cilindro de radio de base 10 dibujándolo con un color verde

grupo3D.getChildren().add(linea);
// Añade la línea al grupo

SubScene subEscena3D = new SubScene(grupo3D, 0, 0, true,
        SceneAntialiasing.BALANCED);
// Añade el grupo a la escena 3D,
// donde se realiza un renderizado con antialiasing

subEscena3D.setFill(new Color(0.2, 0.2, 0.2, 1));
// Establece un color gris de fondo

camara = new PerspectiveCamera(true);
// Crea una cámara para realizar la proyección

camara.setNearClip(0.1);
camara.setFarClip(100000.0);
// Se van a representar en pantalla todos los objetos situados desde una distancia
// de 0.1 unidades de la cámara y hasta una distancia de 100000 unidades

Rotate rotacionXCamara = new Rotate(160, Rotate.X_AXIS);
Rotate rotacionYCamara = new Rotate(-30, Rotate.Y_AXIS);
Translate traslacionCamara = new Translate(0, 0, -3000);
camara.getTransforms().addAll(rotacionXCamara, rotacionYCamara, traslacionCamara);
// A la cámara se le aplican las rotaciones y traslación indicadas por parámetro para
// situarla en una posición inicial. Primero una traslación de -3000 unidades en el
// eje Z, luego una rotación de -30° en el eje Y y finalmente una rotación de 160°
// en el eje X

subEscena3D.setCamera(camara);
// Establece la cámara para la escena 3D

PointLight luzPuntual = new PointLight(new Color(0.6, 0.6, 0.6, 1));
luzPuntual.setTranslateX(10000);
luzPuntual.setTranslateY(20000);
luzPuntual.setTranslateZ(30000);
grupo3D.getChildren().add(luzPuntual);
// Se define en la escena una luz puntual situada en (10000, 20000, 30000) con un
// color gris definido por sus componentes roja = verde = azul = 0.6

grupo3D.getChildren().add(new AmbientLight(new Color(0.3, 0.3, 0.3, 1)));
// Añade una luz difusa de color gris formada por rojo = verde = azul = 0.3


// ---------------- Parte superior de la ventana en 2D -------------------------

Label etiquetaResultado = new Label("");
// Etiqueta para mostrar un texto inicialmente vacío

etiquetaResultado.setPadding(new Insets(5));  // Márgenes de 5 puntos alrededor

ArrayList<String> opcionesComboBox = new ArrayList<String>();
opcionesComboBox.add("A Coruña");
opcionesComboBox.add("Lugo");
opcionesComboBox.add("Ourense");
opcionesComboBox.add("Pontevedra");
// ArrayList de cadenas con el texto de las opciones que aparecen en el ComboBox
```

```
ComboBox comboProvincias = new ComboBox(
        FXCollections.observableArrayList(opcionesComboBox));
// Crea un ComboBox para mostrar esas opciones

comboProvincias.setValue("Pontevedra");
// Establece la opción seleccionada inicialmente

comboProvincias.setOnAction(evento -> {
    // Al método setOnAction se le pasa una expresión Lambda que se ejecutará
    // cuando cambie la opción elegida en el ComboBox

    etiquetaResultado.setText("Provincia: " +
            comboProvincias.getSelectionModel().getSelectedItem());
    // Muestra la opción elegida en la etiqueta
});

CheckBox checkEsfera1 = new CheckBox("Esfera 1");  // CheckBox con texto "Esfera 1"
checkEsfera1.setPadding(new Insets(5));  // Establece márgenes de 5 puntos alrededor
checkEsfera1.setSelected(true);  // Inicialmente el CheckBox está seleccionado

checkEsfera1.setOnAction(evento -> {
    // Al método setOnAction se le pasa una expresión Lambda que se ejecutará
    // cuando cambie el estado de selección del CheckBox

    esfera1.setVisible(checkEsfera1.isSelected());
    // La esfera 1 será visible si el CheckBox está seleccionado
});

CheckBox checkEsfera2 = new CheckBox("Esfera 2");
checkEsfera2.setPadding(new Insets(5));
checkEsfera2.setSelected(true);
checkEsfera2.setOnAction(evento -> {
    esfera2.setVisible(checkEsfera2.isSelected());
});
// CheckBox para controlar la visibilidad de la esfera 2

CheckBox checkEjes = new CheckBox("Ejes");
checkEjes.setPadding(new Insets(5));
checkEjes.setSelected(true);
checkEjes.setOnAction(evento -> {
    grupoEjes.setVisible(checkEjes.isSelected());
});
// CheckBox para controlar la visibilidad de los ejes



ToolBar barraSuperior = new ToolBar();
barraSuperior.getItems().add(checkEjes);
barraSuperior.getItems().add(checkEsfera1);
barraSuperior.getItems().add(checkEsfera2);
barraSuperior.getItems().add(comboProvincias);
barraSuperior.getItems().add(etiquetaResultado);
barraSuperior.setOrientation(Orientation.HORIZONTAL);
// Barra superior horizontal donde se colocan los CheckBox, ComboBox y Label. Es una
// colección donde se guardan objetos de esas clases, que son derivadas de la clase
// base Node


// -------------------------- Ventana --------------------------

// Contenido de la ventana
BorderPane contenidoVentana = new BorderPane();
contenidoVentana.setCenter(subEscena3D);
contenidoVentana.setTop(barraSuperior);
// Crea el contenido de la ventana, situando la subescena 3D en el centro y la barra
// superior arriba

subEscena3D.heightProperty().bind(contenidoVentana.heightProperty());
subEscena3D.widthProperty().bind(contenidoVentana.widthProperty());
// El tamaño de la subescena 3D se adapta a cambios de tamaño en la ventana

Scene escena = new Scene(contenidoVentana);
// Crea una escena a partir del contenido a mostrar
```

```
escena.setOnMousePressed((MouseEvent evento) -> {
    ratonXVentanaAntes = evento.getSceneX();
    ratonYVentanaAntes = evento.getSceneY();
});
// Define mediante una expresión Lambda el código que se ejecuta cuando se produce
// el evento de ratón consistente en la pulsación de un botón del ratón.
// Guarda en las variables ratonXVentanaAntes y ratonYVentanaAntes las coordenadas
// X e Y donde se pulsó el ratón.

escena.setOnScroll((ScrollEvent evento) -> {
    // Define mediante una expresión Lambda el código que se ejecuta cuando el
    // usuario gira la rueda del ratón.

    double factor = 0.02;
    camara.setRotationAxis(Rotate.Z_AXIS);
    double roll = camara.getRotate() + evento.getDeltaY() * factor;
    // Con el método getDeltaY se recoge la cantidad de movimiento en la rueda del
    // ratón, que se multiplica por un factor y se añade al ángulo de rotación en el
    // eje Z de la cámara, que es el eje de visión

    Rotate rotacion = new Rotate(roll, Rotate.Z_AXIS);
    camara.getTransforms().addAll(rotacion);
    // Modifica la rotación de la cámara en su eje Z
});


escena.setOnKeyPressed(event -> {
    // Define la expresión Lambda que se ejecuta cuando se detecta la pulsación
    // (única o repetida) de una tecla

    double desplazamiento = 10;
    if (event.isShiftDown()) {
        desplazamiento = 60;
    }
    // Desplazamiento de la cámara. Con mayúsculas se desplaza el doble

    KeyCode tecla = event.getCode();  // Tecla pulsada

    if (tecla == KeyCode.S) {
        // Si tecla S, desplazamiento hacia atrás, en eje Z de la cámara, que es su
        // eje de visión
        Translate traslacion = new Translate(0, 0, -desplazamiento);
        camara.getTransforms().addAll(traslacion);
    }
    if (tecla == KeyCode.W) {  // Desplazamiento hacia delante
        Translate traslacion = new Translate(0, 0, desplazamiento);
        camara.getTransforms().addAll(traslacion);
    }
    if (tecla == KeyCode.A) {  // Desplazamiento hacia la izquierda
        Translate traslacion = new Translate(-desplazamiento, 0, 0);
        camara.getTransforms().addAll(traslacion);
    }
    if (tecla == KeyCode.D) {  // Desplazamiento hacia la derecha
        Translate traslacion = new Translate(desplazamiento, 0, 0);
        camara.getTransforms().addAll(traslacion);
    }
    if (tecla == KeyCode.E) {  // Desplazamiento hacia arriba
        Translate traslacion = new Translate(0, -desplazamiento, 0);
        camara.getTransforms().addAll(traslacion);
    }
    if (tecla == KeyCode.C) {  // Desplazamiento hacia abajo
        Translate traslacion = new Translate(0, desplazamiento, 0);
        camara.getTransforms().addAll(traslacion);
    }
});


escena.setOnMouseDragged(evento -> {
    // Expresión Lambda que se ejecuta mientras se arrastra el ratón con algún
    // botón pulsado

    double limitePitch = 90;  // Ángulo máximo de cabeceo, positivo o negativo
```

```
        ratonY = evento.getSceneY();
        ratonX = evento.getSceneX();
        // Obtiene la nueva posición del ratón

        double movimientoRatonY = ratonY - ratonYVentanaAntes;
        double movimientoRatonX = ratonX - ratonXVentanaAntes;
        // Cantidad de movimiento del ratón desde la anterior posición

        double factor = 0.1;
        if (evento.isShiftDown())
            factor = 0.3;
        // Factor por el que se multiplican los ángulos de giro, mayor si se pulsa
        // la tecla Shift

        if (evento.isPrimaryButtonDown()) {
            // Se está pulsando el botón izquierdo del ratón ...

            if (ratonY != ratonYVentanaAntes) {
                // Si hubo movimiento vertical del ratón en la ventana ...

                camara.setRotationAxis(Rotate.X_AXIS);
                double pitch = -movimientoRatonY * factor;
                // Ángulo de cabeceo ó pitch en función del movimiento del ratón

                if (pitch > limitePitch)
                    pitch = limitePitch;
                if (pitch < -limitePitch)
                    pitch = -limitePitch;
                // Limita el pitch

                camara.getTransforms().addAll(new Rotate(pitch, Rotate.X_AXIS));
                // Rota la cámara con ese ángulo con respecto a su eje X
            }

            if (ratonX != ratonXVentanaAntes) {  // Hubo movimiento horizontal del ratón

                camara.setRotationAxis(Rotate.Y_AXIS);
                double yaw = movimientoRatonX * factor;  // Calcula el ángulo yaw

                camara.getTransforms().addAll(new Rotate(yaw, Rotate.Y_AXIS));
                // Rota la cámara con ese ángulo con respecto al eje Y de la cámara
            }

        }

        ratonXVentanaAntes = ratonX;
        ratonYVentanaAntes = ratonY;
        // Recuerda la posición del ratón para el siguiente evento
    });


    escena.setOnMouseClicked((MouseEvent evento) -> {
        // Trata con una expresión Lambda el evento producido cuando se hace click
        // con el ratón

        Node captura = evento.getPickResult().getIntersectedNode();
        // Cuando se hace click sobre un elemento gráfico, se obtiene una referencia
        // al objeto correspondiente. Es una referencia de la clase Node, que es clase
        // base para Box, Sphere, etc

        if (captura instanceof Sphere) {
            // Si es una referencia a un objeto de la clase Sphere ...

            Sphere esferaSeleccionada = (Sphere) captura;
            // La convierte a una referencia de la clase Sphere

            if (esferaSeleccionada == esfera1)
                etiquetaResultado.setText("Click en esfera 1");
            else if (esferaSeleccionada == esfera2)
                etiquetaResultado.setText("Click en esfera 2");
            // Muestra un mensaje en la etiqueta indicando sobre qué esfera se hizo click
        }
    });
```

```
        stage.setTitle("Escena 3D");  // Establece el título de la ventana
        stage.setScene(escena);  // Establece la escena que se muestra en la ventana
        stage.setWidth(800);  // Establece ancho inicial de la ventana
        stage.setHeight(600);  // Establece alto inicial de la ventana
        stage.show();
    }  // Fin del método start


    public static void main(String[] args) {
        launch(args);
    }
}
```